

RSX-11M/M-PLUS

Error Logging Manual

Order No. AA-L674B-TC

RSX-11M Version 4.1

RSX-11M-PLUS Version 2.1

digital equipment corporation • maynard, massachusetts

First Printing, January 1982
Revised, April 1983

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Copyright © 1982, 1983 by Digital Equipment Corporation
All Rights Reserved.

Printed in U.S.A.

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	RSX
DEC/CMS	EduSystem	UNIBUS
DEC/MMS	IAS	VAX
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	digital
DECUS	RSTS	
DECwriter		

ZK2344

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed
with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or
approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

CONTENTS

	Page
PREFACE	ix
SUMMARY OF TECHNICAL CHANGES	xi
CHAPTER 1 INTRODUCTION	
1.1 THE PURPOSE OF ERROR LOGGING	1-1
1.2 ERROR LOGGING OPERATION	1-1
1.2.1 Executive Routines	1-3
1.2.2 ERRLOG and ELI	1-4
1.2.3 RPT	1-4
1.2.4 CFL	1-5
1.3 ERROR LOGGING OPTIONS	1-6
1.3.1 Unexpected Traps or Interrupts	1-6
1.3.2 Device Errors	1-6
1.3.3 Interrupt Timeouts	1-6
1.3.4 Memory Errors	1-6
CHAPTER 2 ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)	
2.1 INSTALLING ERRLOG AND ELI	2-1
2.2 USING ERRLOG AND ELI	2-2
2.3 ELI SWITCHES	2-2
2.3.1 Logging Switches	2-4
2.3.2 ERROR Limiting Switches	2-6
2.3.2.1 The Limit Switch	2-6
2.3.2.2 The Hard Limit Switch	2-7
2.3.2.3 The Reset Switch	2-7
2.3.2.4 The Soft Limit Switch	2-7
2.3.3 File Naming Switches	2-8
2.3.3.1 The Log Switch	2-8
2.3.3.2 The Append Switch	2-8
2.3.3.3 The Switch Switch	2-8
2.3.3.4 The Backup Switch	2-9
2.3.4 Display Switch	2-10
2.4 ERRLOG AND ELI MESSAGES	2-11
2.4.1 ELI Messages	2-11
2.4.2 ERRLOG Messages	2-12
CHAPTER 3 REPORT GENERATOR TASK (RPT)	
3.1 INSTALLING AND RUNNING RPT	3-1
3.2 USING RPT TO CREATE ERROR LOG REPORTS	3-2
3.2.1 The RPT Command Line	3-2
3.2.2 Using Multiple Qualifiers in RPT Command Lines	3-3
3.2.3 Using the Default RPT Command Line	3-4
3.3 RPT REPORT SWITCHES	3-5
3.3.1 Packet Selection Switches	3-7
3.3.1.1 The Date Switch	3-7

CONTENTS

Page

3.3.1.2	The Device Switch	3-8
3.3.1.3	The Packet Switch	3-8
3.3.1.4	The Drive and Pack Serial Number Switch . . .	3-9
3.3.1.5	The Type Switch	3-9
3.3.1.6	The Volume Label Switch	3-10
3.3.2	Report Format Switch	3-11
3.3.2.1	Brief Reports	3-11
3.3.2.2	Full Reports	3-15
3.3.2.3	Register Reports	3-18
3.3.2.4	No Report	3-20
3.3.3	Summary Switch (RSX-11M-PLUS only)	3-20
3.3.3.1	The All Qualifier	3-20
3.3.3.2	The Error Qualifier	3-20
3.3.3.3	The Geometry Qualifier	3-23
3.3.3.4	The History Qualifier	3-25
3.3.3.5	The None Qualifier	3-27
3.3.4	The Report Switch	3-27
3.3.4.1	Predefined Switch Strings	3-27
3.3.4.2	User Defined Switch Strings	3-28
3.3.5	The Width Switch	3-28
3.4	ERLCNF REPORT MESSAGES	3-29
3.5	ERLRPT REPORT MESSAGES	3-35

CHAPTER 4 ERROR LOG CONTROL FILE ARCHITECTURE

4.1	TERMS AND CONCEPTS	4-1
4.2	CONTROL FILE MODULE ARCHITECTURE	4-2
4.2.1	RSX-11M and RSX-11M-PLUS Control File Modules .	4-3
4.2.2	Program Control Flow	4-10
4.2.3	Compilation Paths	4-11
4.2.4	Modification and Recompilation	4-13
4.3	INTERNAL INTERFACES	4-13
4.3.1	Interaction Between Dispatcher and Device-Level Modules	4-13
4.3.1.1	Interaction between DSP2M1 and ERM23	4-14
4.3.1.2	Interaction Between DSP2P1 and ERM23	4-15
4.4	DISPATCHING	4-16
4.4.1	Event-Level Dispatching	4-16
4.4.2	Device-Level Dispatching	4-18
4.4.3	CPU-level Dispatching	4-19
4.5	SUPPORT OF NON-DIGITAL DEVICES	4-19
4.5.1	Error-logging of Unknown Devices	4-19
4.5.2	Providing Driver Support for a Non-DIGITAL Device	4-19
4.5.2.1	\$BMSET on RSX-11M	4-20
4.5.2.2	\$BMSET on RSX-11M-PLUS	4-20
4.5.2.3	\$DVTMO and \$DTOER on RSX-11M	4-20
4.5.2.4	\$DVTMO and \$DTOER on RSX-11M-PLUS	4-21
4.5.2.5	\$DVERR and \$DVCER on RSX-11M	4-21
4.5.2.6	\$DVERR and \$DVCER on RSX-11M-PLUS	4-22
4.5.2.7	\$NSIER	4-22
4.5.2.8	\$FNERL	4-22
4.5.2.9	\$LOGGER	4-23
4.5.2.10	LOGTST	4-23
4.5.2.11	\$CRPKT	4-23
4.5.2.12	CALDEV on RSX-11M-PLUS	4-24
4.5.2.13	\$QUPKT	4-24
4.5.2.14	\$QERMV	4-25
4.5.3	Error-Logging Support for a Non-DIGITAL Device	4-25
4.5.3.1	How to Write a Device-Level Module	4-25
4.5.3.1.1	MODULE Statement	4-26
4.5.3.1.2	PROCEDURE Statement	4-26
4.5.3.1.3	SUBPACKET Declaration	4-26
4.5.3.1.4	Register Definitions	4-27

CONTENTS

Page

4.5.3.1.5	Declaration of Local Work Variables and Tables	4-29
4.5.3.1.6	Loading of the Intermodule Variables	4-29
4.5.3.1.7	Determination of the Error Type	4-30
4.5.3.1.8	Coroutine Back to Caller	4-30
4.5.3.1.9	Perform the Bit-To-Text Translation and Register Printing	4-30
4.5.3.1.10	Indicate Any Notes that are Required	4-31
4.5.3.1.11	Exit the module	4-31
4.5.3.2	How to Write a Notes Module	4-31
4.5.3.2.1	MODULE Statement	4-32
4.5.3.2.2	PROCEDURE Statement	4-32
4.5.3.2.3	Notes Heading	4-32
4.5.3.2.4	Selecting a Note for Printing	4-33
4.5.3.2.5	Handling an Unknown Note Number	4-33
4.5.3.2.6	Getting the Next Note	4-33
4.5.3.2.7	Exit the Module	4-33
4.5.3.3	MASSBUS and Non-MASSBUS Considerations	4-33
4.5.3.4	Making the New Device-Level Module Known	4-34
4.6	CODE EXAMPLES	4-37
4.6.1	RM02/03 Device-Level Module ERM23	4-37
4.6.2	DSP2M1 Dispatcher Module for RSX-11M	4-50
4.6.3	DSP2P1 Dispatcher Module for RSX-11M-PLUS	4-57
4.6.4	RM02/03 Notes Module NRM23	4-67
4.6.5	Subpacket Definitions	4-69
4.6.5.1	Subpackets Declared by DISPATCH	4-69
4.6.5.2	Subpackets Declared by DSP1M1/DSP1P1	4-72
4.6.5.3	Subpackets Declared by DSP2M1/DSP2P1	4-73
4.6.5.4	Subpackets Declared by DSP3M1/DSP3P1	4-73
4.6.5.5	Subpackets Declared by DSP4M1/DSP4P1	4-74
4.6.5.6	Subpackets Declared by DSP5M1/DSP5P1	4-74
4.6.5.7	Subpackets Declared by DSP6M1/DSP6P1	4-74
4.6.5.8	Subpackets Declared by DSP7M1/DSP7P1	4-75

CHAPTER 5 CONTROL FILE LANGUAGE GUIDE

5.1	CONTROL FILE OVERVIEW	5-1
5.1.1	Report Generator General Processing	5-1
5.1.2	The General Format of an Error Log Packet	5-2
5.1.3	Control File Language	5-2
5.1.4	General Format of Control File Modules	5-2
5.1.5	Files	5-3
5.2	TYPES AND EXPRESSIONS	5-4
5.2.1	Data Types	5-4
5.2.1.1	LOGICAL Type	5-4
5.2.1.2	STRING Type	5-4
5.2.1.3	ASCII Type	5-5
5.2.1.4	Numeric Types	5-5
5.2.1.5	Field Types	5-7
5.2.1.6	POINTER Type	5-7
5.2.1.7	RSX_TIME Type	5-7
5.2.1.8	VMS_TIME Type	5-7
5.2.2	Variables	5-8
5.2.3	Literals	5-9
5.2.4	Expressions	5-9
5.2.4.1	String Operators	5-9
5.2.4.2	Logical Operators	5-10
5.2.4.3	Relational Operators	5-11
5.2.4.4	Numeric Operators	5-13
5.2.5	Operator Precedence	5-15
5.3	FUNCTIONS	5-16
5.3.1	%CND Functions - Conditional Functions	5-17
5.3.2	%CNV Functions - Conversion Functions	5-17
5.3.2.1	%CNV Functions - Numeric Conversion Functions	5-17

CONTENTS

Page

5.3.2.2	%CNV Functions - Miscellaneous Conversion Functions	5-19
5.3.3	%COD Functions - Encoding Functions	5-19
5.3.4	%COM Functions - Computational Functions	5-20
5.3.5	%CTL Functions - RPT Control	5-21
5.3.6	%LOK Functions - Lookahead Functions	5-21
5.3.7	%PKT Functions - Packet Information	5-22
5.3.8	%RPT Functions - Report Control	5-22
5.3.9	%STR Functions - String Handling	5-23
5.3.10	%TIM Functions - Time Handling	5-25
5.3.11	%USR Function - User I/O Function	5-25
5.4	DECLARATIONS	5-26
5.4.1	Scope of Declarations	5-26
5.4.2	DECLARE Statement	5-26
5.4.3	PACKET Statement	5-28
5.4.4	SUBPACKET Statement	5-29
5.4.5	Conditional Declarations	5-29
5.5	ACTION STATEMENTS	5-30
5.5.1	SET Statement	5-31
5.5.2	INCREMENT and DECREMENT Statements	5-31
5.5.3	WRITE Statement	5-31
5.5.4	WRITE GROUP Statement	5-31
5.5.5	DECODE Statement	5-32
5.6	CONTROL STATEMENTS	5-32
5.6.1	MODULE Statement	5-32
5.6.2	LITERAL Statement	5-33
5.6.3	CALL Statement	5-33
5.6.4	RETURN Statement	5-33
5.6.5	PROCEDURE Statement	5-33
5.6.6	IF-THEN-ELSE Statement	5-34
5.6.7	CASE Statement	5-34
5.6.8	SELECT Statement	5-34
5.6.9	WHILE/UNTIL/DO Statements	5-35
5.6.10	LEAVE Statement	5-35
5.6.11	BEGIN-END Statement	5-36
5.6.12	Lexical Conditionals	5-36
5.7	TABLES	5-36
5.7.1	Table Structure	5-36
5.7.2	TABLE Statement	5-37
5.7.3	DYNAMIC TABLE Statement	5-37
5.7.4	FILE Statement	5-37
5.7.5	POINTER Statement	5-38
5.7.6	FIND Statement	5-39
5.7.7	PUT Statement	5-39
5.8	LISTS	5-39
5.8.1	LIST Statement	5-39
5.8.2	SEARCH Statement	5-40
5.9	SIGNALLING	5-40
5.9.1	Signalling	5-40
5.9.2	ENABLE Statement	5-40
5.9.3	SIGNAL Statement	5-40
5.9.4	SIGNAL STOP Statement	5-41
5.9.5	MESSAGE Statement	5-41
5.9.6	CRASH Statement	5-41
5.10	PRINT FORMATTING	5-41
5.10.1	FORMAT keyword string	5-41
5.10.1.1	Control Directives	5-42
5.10.1.2	Formatting Directives	5-42
5.10.1.3	Data-formatting Directives	5-42
5.11	USER INTERFACE HANDLING	5-43
5.11.1	Overview of User Interface Handling	5-43
5.11.2	Command Mode	5-43
5.11.3	Option Mode	5-43
5.12	ERLCFL REPORT MESSAGES	5-44

CONTENTS

Page

APPENDIX A	TUNING THE ERROR LOGGING UNIVERSAL LIBRARY
APPENDIX B	DRIVE SERIAL NUMBERS
APPENDIX C	ERROR LOG PACKET FORMAT

EXAMPLES

EXAMPLE 2-1	Error Logging Status	2-10
3-1	Error Log Brief Report	3-13
3-2	Error Log Full Report	3-16
3-3	Error Log Register Report	3-19
3-4	Error Summary Report	3-22
3-5	Geometry Summary Report	3-24
3-6	History Summary Report	3-26
A-1	Sample Execution of TUNE.CMD	A-2
C-1	Error Log Packet Format	C-1

FIGURES

FIGURE 1-1	Error Logging System	1-2
4-1	Structure of Error-Logging Packet	4-3
4-2	Compilation Path for RSX-11M Control File Modules	4-11
4-3	Compilation Path for RSX-11M-PLUS Control File Modules	4-12

TABLES

TABLE 2-1	ELI Switches and Subswitches	2-3
2-2	Error Logging Devices	2-5
3-1	RPT File Specification Defaults	3-3
3-2	RPT Report Switches and Subswitches	3-5
4-1	Error Logging Code/Subcode Combinations	4-17
4-2	Event Types, Codes, and Their Dispatcher Modules	4-18
4-3	The DEVICE_INFO Table	4-34
A-1	Modules in ERRLOG.ULB for RSX-11M	A-4
A-2	Modules in ERRLOG.ULB for RSX-11M-PLUS	A-5
B-1	Significant Digits in Drive Serial Numbers	B-1

PREFACE

This manual contains information about operating the RSX-11M/M-PLUS Error Logging System. It explains how the Error Logger collects information on system events and errors and how the Report Generator and Control File produce various kinds of reports on those events and errors. It also includes information on the control file architecture and on how to add user-written modules. The error logging system allows you to monitor the reliability of the hardware on your system and to set error limits and display messages on the console terminal if the number of errors on a device exceeds those limits.

This manual assumes you are familiar with the following documents:

The RSX-11M/M-PLUS MCR Operations Manual

The RSX-11M/M-PLUS Utilities Manual

The RSX-11M-PLUS or RSX-11M System Generation and Installation Guide

The RSX-11M and RSX-11M-PLUS Information Directory and Master Index define the intended readership for each manual in the documentation set and provide a synopsis of each manual's contents. When this manual refers to other documents, consult the appropriate information directory for information about the document.

INTENDED AUDIENCE

This manual is intended for Field Service personnel, system managers, and others responsible for maintaining the integrity of hardware devices connected to an RSX-11M or RSX-11M-PLUS system.

In addition to understanding the RSX-11M or RSX-11M-PLUS operating system and the Error Logging System, you need a thorough knowledge of the hardware devices that the Error Logging System is monitoring. This manual does not attempt to describe or explain the hardware information that appears in the Error Log Reports. For information about a specific device, consult the hardware documentation for that device.

STRUCTURE OF THIS DOCUMENT

Chapter 1 provides an overview of the purpose and function of the Error Logging System. It describes some features and limitations of the system and explains the operating system resources that error logging requires.

Chapter 2 describes the procedures for operating the Error Logger and explains the Error Log Interface commands to control logging and limiting.

Chapter 3 describes the procedures for operating the Report Generator and describes the report formatting available.

Chapter 4 explains the control file modules in detail, including flow of program control, interfaces between modules, and module dispatching. A knowledgeable system programmer can use the information presented to add user-written modules to the Error Logging System. The chapter includes extensively annotated examples of DIGITAL-supplied modules.

Chapter 5 describes the Control File Language, which is used to write control-file modules.

Appendix A describes the indirect command file, TUNE.CMD, that you can use to remove devices from the Error Logging ULB and make it smaller.

Appendix B describes the formats used for drive serial numbers on DIGITAL devices.

Appendix C describes the formats for standard error log subpackets.

CONVENTIONS USED IN THIS DOCUMENT

Examples of Error Log Reports illustrate the operation of the Report Generator. They do not attempt to explain the specific hardware-related events that the reports describe.

Black ink in command line descriptions designates what the computer displays at the terminal.

Red ink designates what the user enters at the terminal.

Square brackets [] enclose the optional parameters for an ELI, RPT, or CFL command.

Uppercase characters in command lines or syntax descriptions indicate required syntax for the command.

Lowercase characters indicate variable parameters that the user selects.

Gray shading in text and examples indicates features that appear only on RSX-11M-PLUS systems.

Pink shading in text and examples indicates features that appear only on RSX-11M systems.

SUMMARY OF TECHNICAL CHANGES

The Error Logging System will now allow the hard and soft error limits to be reached independently. Previously, reaching one of the limits would disable logging of either kind of error on that device. Now, reaching the soft limit will not affect the logging of hard errors, nor will reaching the hard limit affect the logging of soft errors.

Device timeouts are now logged as hard errors if unrecoverable, and as soft errors if recoverable.

When generating a report, RPT looks first for LX:[1,6]ERRLOG.ULB. If it fails to find that file, it looks for LB:[1,6]ERRLOG.ULB.

The Executive ERROR module now resides in a directive common on RSX-11M-PLUS systems and may reside in a directive common on RSX-11M systems. Therefore, drivers that create data areas containing information to be passed to the Executive ERROR module must not create the data area in memory mapped by APR5.

There are no user interface changes except for a number of new error messages.

Two new chapters in this manual document the architecture of error log control files and the Control file Language (CFL). These chapters replace Appendix A in the previous version of this manual.

There have been a number of minor changes in CFL. Here is a list of differences between CFL in RSX-11M V4.1 and RSX-11M-PLUS V2.1 and the previous releases:

- **DYNAMIC_TABLE statement:**

DYNAMIC_TABLE is a synonym for FILE. You should use this new statement in place of any FILE statement in new code.

- **FILE statement:**

The FILE statement will be removed from a future release. Please convert your code to use DYNAMIC_TABLE instead of FILE.

- **%CNV\$xxx functions:**

The field_width parameter is now optional and interacts with the optional fill_character parameter to determine whether the resulting string is printed as is or is left- or right-justified. In the earlier version, the digits in the string were always right-justified and blank-filled if no fill_character was specified.

SUMMARY OF TECHNICAL CHANGES

- **%LOK\$LENGTH function:**

This function always returns the length of the data in a packet or subpacket. The length word for the packet or subpacket is not considered part of the data and is not counted in determining the length value returned.

- **%LOK\$BYTE, %LOK\$WORD, %LOK\$LONGWORD functions:**

The offset parameter is the offset within the data of the packet/subpacket at which the byte, word, or longword begins. The offset unit is always in bytes, with the first byte of data in the packet/subpacket being offset 0.

- **%STR\$UPCASE function:**

STR\$UPCASE accepts an ASCII string as a parameter, and returns the ASCII string with all lowercase ASCII characters converted to uppercase.

- **WRITE and WRITE_GROUP statements:**

Because of overlay restrictions, the following operators and functions cannot be used in expressions in WRITE or WRITE_GROUP statements:

- single and double operand numeric operators
- the MATCH operator
- %COD\$xxx functions
- %CTL\$xxx functions
- %PKT\$xxx functions
- %RPT\$xxx functions
- %STR\$xxx functions
- %TIM\$xxx functions
- %USR\$xxx functions

CHAPTER 1

INTRODUCTION

1.1 THE PURPOSE OF ERROR LOGGING

The RSX-11M/M-PLUS Error Logging System records information about errors and events that occur on your system hardware, either for immediate action or for later analysis and reporting. Error logging handles mass storage device (disk and tape) errors, as well as memory errors. Since error logging is a part of the RSX-11M/M-PLUS system, it is most effective for hardware errors that allow the system to continue functioning.

Error logging is not used to detect information about operating system failures or about device problems that cause the system to fail. However, it does provide information about what I/O activities occurred on a device at the time of an I/O failure. If your system includes the Crash Dump Analyzer (CDA), CDA can provide reports on operating system failures.

You can use Error Log Reports to determine that a device is having problems before the device actually fails and causes you to lose data. For example, a report showing a pattern of recurring errors from different blocks on a single disk head may indicate that the head needs to be replaced.

1.2 ERROR LOGGING OPERATION

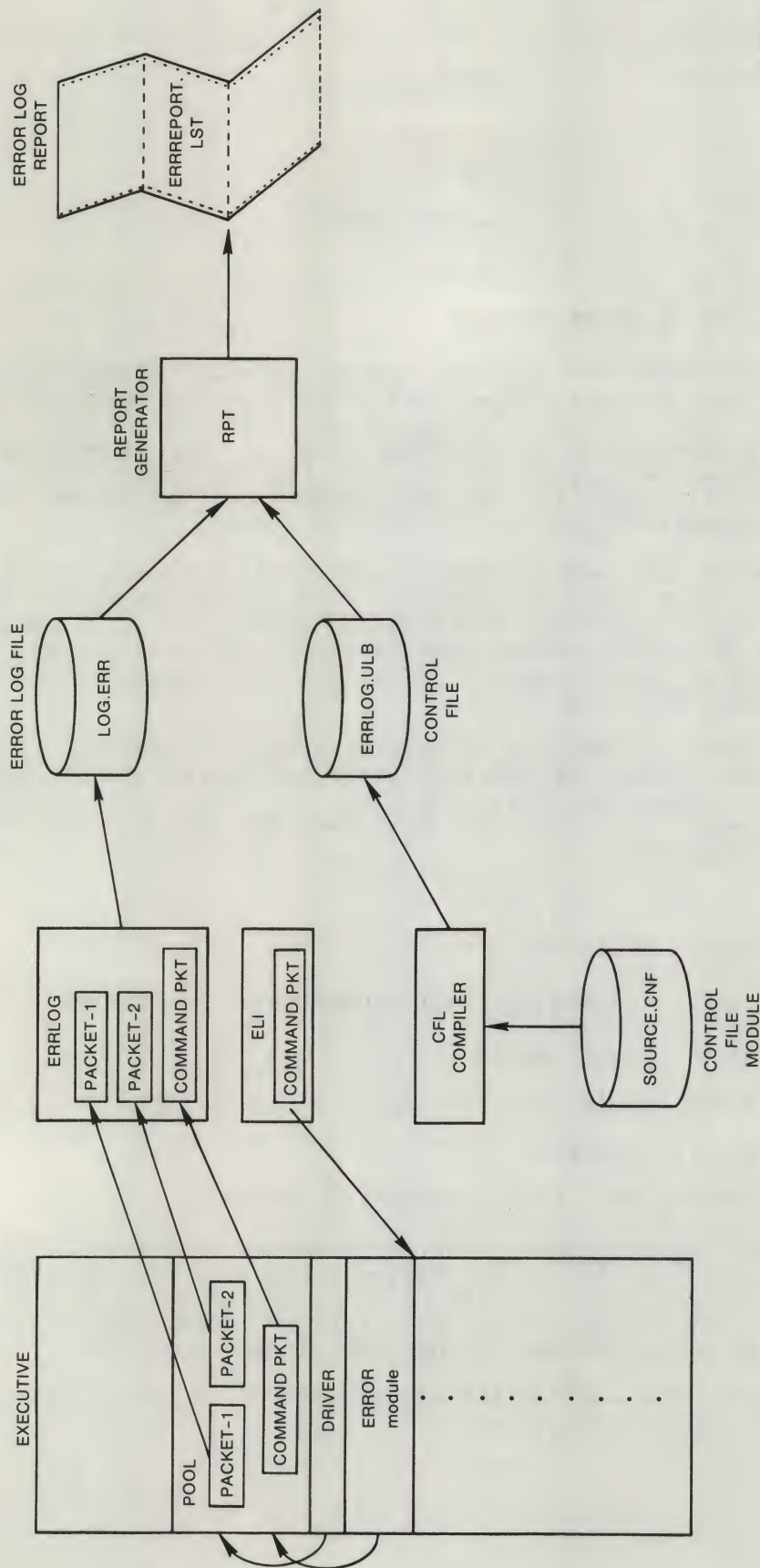
The complete Error Logging System is composed of four tasks.

- The Error Logger (ERRLOG)
- The Error Log Interface (ELI)
- The Report Generator (RPT)
- The Control File Language Compiler (CFL)

When the executive or a device driver detects an error, Executive routines create an Error Log Packet in pool to describe the event. (See Appendix C for a description of the Error Log packet.) ERRLOG then writes the packet from pool into the Error Log File on disk, usually within a few seconds of when the packet is created.

Figure 1-1 shows the interaction of the Error Logging System tasks with routines in the Executive.

INTRODUCTION



ZK-495-81

Figure 1-1 Error Logging System

INTRODUCTION

ERRLOG receives user commands from the Error Log Interface (ELI) to control ERRLOG operation. These commands send error log packets called command packets to the ERRLOG task.

The Report Generator (RPT) generates reports from the information in the Error Log File.

RPT uses a library of modules written in the Control File Language (CFL) to interpret data from the Error Log File and from user commands. The CFL compiler is also part of the Error Logging System. You can use CFL to recompile DIGITAL-supplied Control File Modules to include patches to the modules supplied in the future. You can also use CFL to create and compile Control File Modules for devices other than those DIGITAL supplies. Chapter 4 explains the control file module architecture and includes annotated DIGITAL control file modules. Chapter 5 documents the Control File Language (CFL).

1.2.1 Executive Routines

Whenever the RSX-11M or RSX-11M-PLUS system is running and error logging is active, routines in the Executive collect information from device drivers and other tasks and write the information into error log packets in system pool.

The Executive gathers information on the state of the registers when a device error occurs, and includes information on system events, such as device Mounts and Dismounts. You can also insert a text message into the error log file using the MCR System Service Message command (SSM). (See the RSX-11M/M-PLUS MCR Operations Manual).

If Error Logging is not active on the system, the device drivers still detect each hardware error, but the Executive does not create Error Log packets.

The Error Logging System makes a distinction between hard errors and soft errors. Hard errors are those that cause an I/O operation to be aborted because the device driver cannot recover from the error. The task that issued the I/O request receives an error code indicating that the operation failed. Soft errors are those from which the device driver can recover. The task that issued the I/O request does not receive an error notification because the request eventually succeeds.

The Error Logging System logs both hard and soft errors. Thus, you can have a system functioning properly, with no errors reported to any tasks in the system, with errors still being encountered and logged. Thus error logging terminology sometimes refers to errors as events: they do not always mean an actual failure.

When Error Logging is active, the Executive writes the data from a single event into one Error Log Packet and assigns a sequence number, unique to that event, to the packet. The Resource Monitoring Display (RMD) shows the highest assigned sequence number as ERRSEQ, the total number of errors since error logging operations began.

When ERRLOG writes the packet in a file, the packet gets a number that describes its location in the file relative to other packets. RPT uses this number to refer to the event in later operations. The number does not change unless the organization of the file changes. For example, if an earlier error log file is appended to the current error log file, the packet numbers in the appended file will change.

INTRODUCTION

Thus, you can generate a brief format RPT report to determine the packet numbers of the most significant errors on your system, and then generate a full format report, by packet number, of only those errors.

The Executive includes a directive for error logging (MSG\$) that sends Error Log Packets directly to the Error Logger. (See the RSX-11M/M-PLUS Executive Reference Manual for an explanation of how to use this directive.) User tasks can use MSG\$ to communicate with the Error Logger.

1.2.2 ERRLOG and ELI

ERRLOG writes the Error Log Packets from pool to the Error Log File in binary format. Only RPT can interpret and format data from the Error Log File.

To issue a command to ERRLOG, type an ELI command to perform one of the ERRLOG functions (logging, limiting, or file naming). ELI sends an error log command packet to ERRLOG with instructions on the function to be performed, and ERRLOG returns the results, if any, to ELI.

The ERRLOG task allows you to specify two files to contain the error log packets written to disk. ERRLOG uses the first file, the error log file, unless an error is detected while ERRLOG writes to the file. If an error is detected, ERRLOG switches to the second file, the backup file. ELI commands allow you to establish or change the names of the error log file and backup file.

The error logging system automatically limits the number of events it logs on a given device. This error limit can be changed dynamically by ELI commands while error logging is running. The system does limiting in case the device starts to accumulate a large number of errors. Without limiting in these cases, the error log file would quickly become large and difficult to analyze. The limiting does not throw away useful information, because usually when a large number of events occurs on a device, most of them are the same and you can generalize from a report on a small number of the events.

After a device reaches a particular error limit, logging of that type of error on the device stops until you reset the error count to zero or raise the error limit.

ERRLOG sends a message to the console terminal or to any terminal that has allocated the device, explaining that the device reached the error limit. Limiting does not affect operation of the device itself; it only starts or stops error logging on the device.

1.2.3 RPT

RPT creates reports on the data in the Error Log File, based on information in the Error Log Control File and commands supplied by the user. Modules in the Error Log Control File tell RPT how to interpret and print entries from the Error Log File for a specific operating system.

When you are ready to generate an Error Log Report, you can run RPT to select the information you want to include in the report. RPT can generate reports in brief and full format on any collection of Error Log Packets you select. For example, you can select reports on a

INTRODUCTION

specific device by device name, device type, volume label, pack identification, or drive serial number. You can also select reports of a specific error type or you can select a full report of all the Error Log Packets in the Error Log File.

Error log reports can contain both context information and device-supplied information.

Context information, which appears in full format reports, contains operating system version information and some information about the CPU model. Context information on the I/O operation that encountered the failure is recorded for device errors. This information is useful to correlate events recorded in the error log file with other events in the system. For example, hard I/O errors often cause the task issuing the I/O request to exit with an error, since many tasks cannot recover from I/O errors. Information on the I/O operation is also useful to determine the operation the device driver attempted at the time of the failure.

Error Logging on RSX-11M-PLUS systems records information on each I/O operation taking place on other system error logging devices when an error occurs. This information is useful if you have a failure that may be related to interactions between devices. For example, I/O bus timing problems may show up as a problem only when more than one I/O operation takes place at the same time on the system.

This support is optional on RSX-11M systems.

In a full report, RPT also includes all device-supplied information, including registers and any other information the device provides. Each device supplies one or more machine words of information when an error occurs. RPT decodes each item of device information according to the terminology used in the device maintenance manual. If additional information is useful to understand the significance of a decoded item, that information is listed in parentheses.

Decoded items that are abnormal are flagged with a "*" in reports. These items may or may not represent error conditions, depending on the state of the device. Interpret items flagged with a "*" as "look at me first". RPT reports flag more than one item on most devices.

RPT reports also flag more than one item if a device encounters an error or cannot perform an operation because of another error condition. This condition occurs when an abnormal device status condition causes an I/O function to fail. The RPT report flags both the I/O function failure and the abnormal device status.

An error type definition in the RPT report then boils all the device-supplied information down to a single item reflecting the most probable error reported by the device.

1.2.4 CFL

The Error Logging System includes a Control File Language compiler (CFL) used to recompile patched DIGITAL-written Control File Modules or user-written modifications or additions to modules. Chapters 4 and 5 describe the operation of the CFL compiler and the DIGITAL-supplied control file module for the RM02/RM03.

INTRODUCTION

1.3 ERROR LOGGING OPTIONS

Routines in the Executive respond to four types of errors:

- Unexpected traps or interrupts
- Device errors
- Device timeouts
- Memory errors

All systems that include Error Logging support at system generation include support for logging the first three types of errors. RSX-11M-PLUS also includes support for logging memory errors. However, support for logging memory errors is a separate system generation option on RSX-11M.

1.3.1 Unexpected Traps or Interrupts

When your system includes Error Logging support, all unused system vectors are filled with pointers to routines in the Executive. Therefore, routines in the Executive are called if a trap or interrupt occurs to one of these unused vectors. For example, a noisy electrical environment or a static discharge may cause an unexpected trap or interrupt to one of the unused vectors, or a valid interrupt may be vectored to the wrong address. In these cases, the Executive records this information.

1.3.2 Device Errors

Device errors are problems that a device encounters while carrying out a software-requested operation. When a device error occurs and Error Logging is active, the device driver calls Executive routines to record the contents of the device registers or other hardware-supplied information. The registers indicate the state of the device and its controller. The routines also record information found in the actual I/O request to the driver, such as the type of operation attempted. This information aids you in the interpretation of the device error.

1.3.3 Interrupt Timeouts

Interrupt timeouts occur when the device that initiated an operation fails to complete the operation within the length of time the driver specified. Software timers that start when the transfer starts, detect interrupt timeouts. The system records the same information for timeouts that it records for device errors.

1.3.4 Memory Errors

Memory errors occur when the parity bit stored with the data during a write operation does not match the parity calculated when the data is read. Some types of main memory use parity to ensure integrity of the information. All RSX-11M-PLUS systems except the pregenerated systems include support for logging memory parity errors. The support is a system generation option on RSX-11M.

CHAPTER 2

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

This chapter describes how to use the Error Log Task (ERRLOG) and the Error Log Interface (ELI). Chapter 1 provided a general overview of how ERRLOG and ELI work, along with the Report Generator (RPT), to form the complete Error Logging System.

ERRLOG gets event and status information from device drivers and the executive in the form of Error Log Packets and writes the packets in an Error Log File on disk. The executive also performs error limiting to allow a maximum number of errors to be logged on each device before logging stops.

ELI, the user interface to ERRLOG, includes switches to:

- Start or stop logging or limiting
- Change device error limits or error counts
- Establish or change log file or backup file names
- Display information about the error logging status of any device or of the entire system

ERRLOG is the only part of the Error Logging System that must be installed for error logging to occur. You can install ELI when you issue commands to ERRLOG and install RPT when you create reports.

2.1 INSTALLING ERRLOG AND ELI

To install the ERRLOG task, enter the following MCR command from a privileged terminal or as an entry in the system startup command file:

```
INS $ERL(RET)
```

To install ELI, enter the following MCR command from a privileged terminal or as an entry in the system startup command file:

```
INS $ELI(RET)
```

If ELI is not installed, you can invoke it from a privileged terminal using the following MCR command:

```
RUN $ELI(RET)
```

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

2.2 USING ERRLOG AND ELI

To invoke ELI after it is installed, issue the following MCR command from any terminal:

```
ELI (RET)
ELI>
```

You can use the ELI /SH switch to display error logging information from any terminal. However, you must use a privileged terminal to execute any other ELI commands.

Enter each command on a separate line unless the command description specifies otherwise.

The general format of an ELI command is:

```
[filespec]/switch1[...switchn]
```

filespec

A device mnemonic or the name of an error log file, backup file, or file to append to the current error log file.

switches

Switches to set, change, or display ERRLOG operation. (You must specify at least one switch on each ELI command line.)

If you want to use only the ERRLOG defaults and start logging, enter the following ELI command:

```
/LOG (RET)
```

This command starts ERRLOG, using LB:[1,6]LOG.ERR as the default log file and LB:[1,6]BACKUP.ERR as the default backup file. You must specify the /LOG switch to use ERRLOG defaults.

The /LOG switch also starts error limiting to limit the number of hard and soft errors ERRLOG records on each device before it stops logging on that device. The default error limit, used when you begin limiting with the /LOG switch, is five hard errors and eight soft errors for each device. You can change these limits with the /HL or /SL switches described in Section 2.3.2. However, you cannot use the switches to change limits on the same command line as the /LOG switch.

2.3 ELI SWITCHES

This section describes the ELI switches and subswitches, divided into four types:

- Logging switches
- Limiting switches
- File naming switches
- Display switch

Remember that these switches only control operation of the Error Logger. Chapter 3 describes the RPT commands that generate actual Error Log Reports. Chapter 5 describes the commands that control the Control File Language Compiler.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

Table 2-1 summarizes the ELI switches in alphabetical order. ELI syntax requires that you specify at least two characters of a switch name and as many additional characters as it takes to make the switch unique. However, the Logging and Limiting switches are called /LOG and /LIM to make their names easier to remember.

Table 2-1
ELI Switches and Subswitches

Switch	Subswitch	Function
filespec/AP (Append)		Appends the specified file to the current Error Log File.
	/DE (Delete)	Deletes the specified file after appending it to the current Error Log File.
filespec/BA (Backup)		Sets the name for a backup file to the next highest version of the file named.
device(s)/HL:n (Hard Error Limit)		Set limits for hard (unrecoverable) errors on a device. You can use /SL, the Soft Error Limits switch, on the same command line.
/LIM (Limiting)		Starts the use of error limiting, using either default limits or those set with ELI switches. The /LOG switch begins error limiting by default.
/-LIM /NOLIM (No Limiting)		Stops the use of error limiting.
[filespec]/LOG (Logging)		Begins error logger operation, turns on error limiting by default, and, if you specify a file name, overrides the default name of the error log file (LB:[1,6]LOG.ERR). If the error log file already exists, the /LOG switch uses the existing file.
	/-LIM	Turns off error limiting while the error logger is running.
	/NV (New Version)	Creates a new version of the given file instead of using the current version.
/-LOG /NOLOG (No Logging)		Stops error logger operation and turns off error limiting.

(continued on next page)

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

Table 2-1 (Cont.)
ELI Switches and Subswitches

Switch	Subswitch	Function
device(s)/RE (Reset)		Resets the QIO and error counts on the specified devices to zero.
device(s)/SH (Show)		Displays error logging information for the specified devices. (If you do not specify device names, /SH displays information for all error logging devices on the system.)
device(s)/SL:n (Soft Error Limit)		Sets limits for soft (recoverable) errors on a device. (You can use /HL, the Hard Error Limit switch, on the same command line.)
filespec/SW (Switch)		Copies current error log file to the specified file and transfers logging to that file.
	/DE	Deletes the old file after the /SW switch performs the copy operation.
	/NV	Creates a new version of the specified file instead of appending data to the current version.

2.3.1 Logging Switches

[filespec]/LOG

/-LOG

/NOLOG

ELI Logging Switches start or stop logging on all error logging devices in the system. (See Table 2-2)

Table 2-2 lists the device modules included in the original LB:[1,6]ERRLOG.ULB as distributed with the Error Logging System. However, if you have deleted any device modules from this ULB, using the indirect command file described in Appendix A, your system will not include support for those devices. If you want error logging support for the devices listed in Table 2-2, the Control File Module listed with the device must be included in the ULB. See Appendix A for information on how to include and delete modules from the ULB.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

Table 2-2
Error Logging Devices

Device	Control File Module
ML11	EML11
RK03/RK05	ERK05
RK06/RK07	ERK67
RL01/RL02	ERL12
RM05	ERM05
RM02/RM03	ERM23
RA80/RA81	MSCP80
	MSCPAT
	MSCPCE
	MSCPEN
	MSCPTO
	DEVUDA
RA60	MSCP60
	MSCPAT
	MSCPCE
	MSCPEN
	MSCPTO
	DEVUDA
RC25/RD51/RX50	MSCPSD
	MSCPAT
	MSCPCE
	MSCPEN
	MSCPTO
	DEVUDA
RM80	ERM80
RP07	ERP07
RP02/RP03	ERP23
RP04/RP05/RP06	ERP456
RS11	ERS11
RS03/RS04	ERS34
RX01	ERX01
RX02	ERX02
TA11	ETA11
TC11	ETC11
TS11/TU80	ETS11
TU58	ETU58
TU77	ETU77
TU16/TE16/TU45	ET1645
TU60	ETU60
TS03/TE10/TU10	ET0310
TSV05	ETSV05

The /LOG switch begins error logging operation and optionally allows you to specify a file in which the error logger writes the data it collects. (See the file naming section below.) If you specify an existing file, the /LOG switch appends new data to that file unless you also specify the New Version switch (/NV) in the command line.

The /LOG switch also turns on error limiting, by default, unless you specify the No Limiting (/LIM) switch to override it.

The NOLOG (/NOLOG) switch stops error logging and, by default, stops error limiting.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

/LOG Subswitches:

You can use the following subswitches on a command line with the /LOG switch:

/-LIM[IT]

The /-LIM subswitch turns off error limiting. This subswitch overrides the default ERRLOG operation in which /LOG automatically turns on error limiting.

/NV

The /NV subswitch causes the error logger to create a new version of the error log file (either the file you specify in the command line or the default error log file LB:[1,6]LOG.ERR). This subswitch overrides the default operation in which the /LOG switch appends data to the current version of the error log file.

2.3.2 ERROR Limiting Switches

The following switches control the error limiting operation of ERRLOG. You can use them to start or stop error limiting or to change error limits on specific devices. When a device reaches the user-specified error limit or the default error limit, ERRLOG displays the following warning message on the console terminal or on any terminal that has allocated or attached the device:

```
ERRLOG -- **WARNING: Device dd: Exceeded (xxxx) Limit (n)
```

In the message, xxxx is the type of limit (hard or soft) and n is the number to which the limit is set.

When the device reaches an error limit, error logging for that type of error stops on the device until you reset the error and QIO counts to 0 or raise the error limit.

You can reset the error and QIO counts to zero with the ELI /RE switch. Mounting or dismounting the device or rebooting the system also resets the error and QIO counts to zero. However, using the /-LOG switch to stop logging does not reset the error and QIO counts.

Logging on a device stops only when the device reaches both of the limits set for hard and soft errors. If, for example, the device reaches its limits for hard errors but not for soft errors, it will continue to log soft errors until the soft error limit is also reached.

2.3.2.1 The Limit Switch

/LIM

/-LIM

/NOLIM

The /LIM switch starts or stops use of error limits. These limits are set by default for all devices on the system when you enable error logging or they are set for individual devices with the hard and soft limit switches described below. The /LIM switch does not activate error logging if it is not currently active on the system.

When you specify the /LOG switch to begin error logging, it automatically starts error limiting on all error logging devices unless you inhibit limiting with the /-LIM switch.

2.3.2.2 The Hard Limit Switch

```
devl:[,...devn]:/HL:n
```

The /HL switch sets limits for the number of hard errors that error logging records on the device specified. Hard errors occur on a device when an I/O operation fails and cannot be recovered by the device driver. You can set hard error limits for more than one device in the same command line, as long as the limits are the same. The default hard error limit on each device is five.

The value n can be 0 to 255. If you set the limit to 255, logging continues without stopping (the limit is infinite). If the limit is set to 0, no errors will be logged.

Subswitch:

You can use the following switch as a subswitch on a command line with the /HL switch:

```
/SL:n
```

In this way, you can set both hard and soft error limits for devices on the same command line.

2.3.2.3 The Reset Switch

```
devl:[,...devn:]/RE[SET]
```

The /RE switch resets the QIO count and error count for the specified devices to zero. You can specify up to 14 devices in one command line. You cannot reset QIO and error counts on all devices in the system at once by specifying the /RE switch without specifying devices.

When ERRLOG resets the counts to zero, it displays the following message on the Console Terminal:

```
ERRLOG -- Error and QIO counts reset for ddnn:
```

2.3.2.4 The Soft Limit Switch

```
devl:[,...devn:]/SL:n
```

The /SL switch sets limits for soft errors. Soft errors occur on a device when an I/O operation fails, but succeeds in a subsequent retry attempt. You can set soft error limits for more than one device in the same command line, as long as the limit is the same. The default soft error limit for each device is eight.

The value n can be 0 to 255. If you set the limit to 255, logging continues without stopping (the limit is infinite). If the limit is set to 0, no errors will be logged.

Subswitch:

You can use the following subswitch on a command line with the /SL switch:

/HL

In this way, you can set both hard and soft error limits for devices on the same command string.

2.3.3 File Naming Switches

The following sections describe switches that establish and change the names of Error Log Files and Backup Files.

2.3.3.1 The Log Switch

[filespec]/LOG

The /LOG switch, which also initializes the error logger, sets the name of the error log file that the error logger uses. If you specify an existing error log file, the default operation is to append data to the current version of that file. To override the default, specify the /NV switch. The error logger then creates and writes data in a new version of the file. This switch does not work when error logging is already active on your system. The default error log file specification is LB:[1,6]LOG.ERR. The /LOG switch also specifies LB:[1,6]BACKUP.ERR as the backup file. See Section 2.3.3.4 for more information.

2.3.3.2 The Append Switch

filespec/AP[PEND]

The /AP switch appends the specified file to the end of the current log file. Error logging must be active for this switch to work.

The default operation is to append the specified file to the current error log file and to keep the appended file.

Subswitch:

You can use the following subswitch on the command line with the /AP switch:

/DE[LETE]

The /DE subswitch causes the error logger to delete the specified file after it copies the file to the end of the current error log file.

2.3.3.3 The Switch Switch

[filespec]/SW[ITCH]

The /SW switch copies the current error log file to the file you specify and begins logging in that file. The default operation

appends data to an existing version of the file and preserves the old version of the error log file.

Subswitches:

You can use the following subswitches on the command line with the /SW switch:

/NV

The /NV subswitch creates a new version of the file you specify. This subswitch overrides the default operation in which the /SW switch appends data to the latest version of the file.

/DE[LETE]

The /DE subswitch causes the error logger to delete the current error log file after it copies the file to the new file you specify.

2.3.3.4 The Backup Switch

filespec/BA[CKUP]

The /BA switch specifies the file to be used as a backup file if the Error Logger cannot write to the current log file. By default, the backup file is LB:[1,6]BACKUP.ERR.

The backup file specification is kept, but no file is created until needed. You may wish to have your backup file on a different device from the current log file. By default, both files are on pseudo device LB:.

When the Error Logger cannot write to the current log file, it creates and opens the backup file and writes to it. At that point, you no longer have a backup file, and the Error Logger displays the following message on the Console Terminal:

```
ERRLOG -- Log file error - logging continuing on backup file
```

After error logging switches to the backup file, there is no longer a backup file available.

The error logger uses the specified backup file as the current error log file. It does not rename the file to LOG.ERR, even though the file is now the error log file.

At this point, you should specify a new backup file, using the /BA switch. Otherwise, if error logging cannot write to the new log file, it will not be able to continue by writing in a backup file.

If the error logger tries to switch logging to a nonexistent backup file, it displays the following message:

```
ERRLOG -- Backup file error - logging discontinued
```

When that happens, logging stops and must be restarted.

If you create the backup file on a disk other than the disk containing the error log file, this ensures that logging will continue even if the disk with the error log file develops problems.

2.3.4 Display Switch

The /SH switch allows you to display information on the status of error logging on the system.

```
[dev1,...devn]/SH[OW]
```

```
/SH[OW]
```

The /SH switch allows you to display information on the status of error logging on the system. The /SH switch displays Error Logging information on the devices specified (up to 14). If the command does not specify devices, the Error Logger displays information on all error logging devices in the system. Example 2-1 illustrates the output from the operation of the /SH switch:

Example 2-1 Error Logging Status

```
Error Logging Status      12-JAN-82 00:51:54
```

```
Logging: On      Limiting: On
```

```
Log File: LB:[1,6]LOG.ERR      File ID: DR3: 32,252
```

```
Backup File: LB:[1,6]BACKUP.ERR
```

Device Name	Hard Error Count/Limit	Soft Error Count/Limit	QIO Count	
MM0:	0./5.	0./8.	23.	
MM1:	0./5.	0./8.	9776.	
MM2:	0./5.	0./8.	0.	
MM3:	0./5.	0./8.	0.	
DB0:	0./5.	0./8.	14144.	
DB1:	0./5.	0./8.	0.	
DB2:	0./5.	* 8./8.	46528.	
DR0:	0./5.	0./8.	0.	
DR1:	0./5.	0./8.	0.	
DR2:	0./5.	0./8.	164234.	
DR3:	0./5.	0./8.	625364.	
DS0:	0./5.	0./8.	130.	
DS1:	0./0.	0./0.	0.	(Offline)
DK0:	0./5.	0./8.	1.	
DK1:	0./5.	0./8.	0.	
DM0:	0./5.	0./8.	0.	
DM1:	0./5.	0./8.	0.	
DL0:	0./5.	0./8.	0.	
DL1:	0./5.	0./8.	0.	
DT0:	0./5.	0./8.	0.	
DT1:	0./5.	0./8.	0.	
DT2:	0./5.	0./8.	0.	
DT3:	0./5.	0./8.	0.	
DY0:	0./5.	0./8.	1.	
DY1:	0./5.	0./8.	1.	
DD0:	0./5.	0./8.	0.	
DD1:	0./5.	0./8.	0.	

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

If you specify device names in the /SH switch, the output is the same as Example 2-1, except that the display only includes information on the devices you specified.

The asterisk next to the soft error limit for DB2: indicates that DB2: reached the soft error limit and logging of soft errors stopped. Note that the logging of hard errors will continue on DB2: until the hard error limit is reached.

The display continues to record additional QIOs on the device, even after logging stops because the Executive maintains the QIO count.

Therefore, the ratio of errors to QIOs on the device does not necessarily give you a statistical error percentage.

2.4 ERRLOG AND ELI MESSAGES

ERRLOG displays messages on the console terminal when errors occur during an operation. In some cases, ERRLOG displays messages on any terminal that has allocated or attached the device on which the error occurs. ELI displays messages on the terminal that invoked it. This section describes the messages, their causes, and possible user response.

2.4.1 ELI Messages

ELI -- ERRLOG not installed

Explanation: ERRLOG is not installed on the system.

User Action: Install ERRLOG from a privileged terminal and issue the ELI command again.

ELI -- Failed to communicate with ERRLOG

Explanation: ELI could not communicate with ERRLOG using the Executive directive (MSG\$).

User Action: Fatal error. No user action is possible.

ELI -- File name must be specified

Explanation: You used a Backup, Append, or Switch switch without specifying a file name.

User Action: Reenter the ELI command with an appropriate file specification.

ELI -- Get Command Line error

Explanation: The Get Command Line procedure failed.

User Action: This may be a temporary condition. Retry the operation.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

ELI -- Illegal switch combination

Explanation: You used an ELI switch in combination with subswitches other than those allowed on a command string with that switch. (See Table 3-1.)

User Action: Reenter the command string, specifying a legal combination of switches on each string. Use a separate command string for additional switches, if necessary.

ELI -- Maximum number of devices exceeded

Explanation: You attempted to reset QIO and error counts on more than 14 devices in one command string.

User Action: Specify the /Reset Switch again, with 14 devices or less.

ELI -- Switch requires device name (ddnn:) only

Explanation: You specified both a device name and UFD and/or file name an ELI switch that only accepts a device name.

User Action: Reenter the command; omit the UFD and file name.

ELI -- Syntax error

Explanation: You used an illegal switch or file specification or made some other syntactical error.

User Action: Reenter the command, using the proper command string syntax.

2.4.2 ERRLOG Messages

ERRLOG -- Backup file error - logging discontinued

Explanation: ERRLOG encountered an error when it wrote in the log file. It then tried to write in the backup file, but could not. This error occurs if you fail to establish a new backup file after ERRLOG switches logging to the backup file.

User Action: Issue an ELI /BA command to establish a new backup file and restart logging.

ERRLOG -- Device not in system

Explanation: ERRLOG tried to use a device that is not in the system configuration.

User Action: Check to be sure you specified the correct device and reenter the command. If the device is correct, no user action is possible.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

ERRLOG -- Error and QIO counts reset for ddnn:

Explanation: The error and QIO counts for a given device were reset.

User Action: No user action is necessary. This is an informational message.

ERRLOG -- Error Log packet too long

Explanation: ERRLOG encountered an Error Log Packet that was too large. The error log packet was corrupt.

User Action: If the Error Logging System includes user-generated error log packets, check the code to make sure none of the packets are too long. Otherwise, submit an SPR.

ERRLOG -- Failed to assign LUN

Explanation: ERRLOG tried to assign a Logical Unit Number to a terminal to send a notification message and the assignment failed. This occurs when a device exceeds the error limit set for it and ERRLOG tries to notify the terminal or task that has the device allocated or attached.

User Action: No user action is necessary. The limiting operation succeeded. This informational message tells you ERRLOG was unable to notify the allocating terminal.

ERRLOG -- File I/O error

Explanation: ERRLOG tried to execute a Switch or Append command and could not open the new file or copy the old file to the new one. When this error occurs, logging continues in the original log file.

User Action: No action is required to continue logging. Retry the Switch or Append command.

ERRLOG -- Log file error - logging continued on backup file

Explanation: An error occurred when ERRLOG tried to write in the Error Log File. The logging operation transferred to write in the backup file. The backup file becomes the log file, but retains the given name.

User Action: Issue an ELI command to establish a new backup file. Otherwise, if ERRLOG gets an error when it writes in the new file (the previous backup file), it will not find a backup file to use.

ERRLOG -- Logging already active

Explanation: ERRLOG received an ELI command to begin logging when logging was running.

User Action: No user action is necessary to continue logging.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

ERRLOG -- Logging initialized

Explanation: When ELI starts ERRLOG operation, using the /LOG switch, ERRLOG displays this message on the Console Terminal.

User Action: No user action is necessary. This is an informational message.

ERRLOG -- Logging not active

Explanation: The ERRLOG task is not currently running on your system.

User Action: Issue an ELI /LOG command from a privileged terminal and retry the operation.

ERRLOG -- Logging stopped

Explanation: When ELI stops ERRLOG operation, using the /-LOG switch, ERRLOG displays this message on the Console Terminal.

User Action: No user action is necessary. This is an informational message.

ERRLOG -- No data subpacket

Explanation: ERRLOG tried to use a corrupted data subpacket.

User Action: If the Error Logging System includes a user-written control file module to generate error log packets, check the code. Otherwise, submit an SPR.

ERRLOG -- No device subpacket

Explanation: ERRLOG tried to use a corrupted device subpacket.

User Action: If the Error Logging System includes a user-written control file module to generate error log packets, check the code. Otherwise, submit an SPR.

ERRLOG -- Privilege violation

Explanation: You tried to issue a privileged ELI command (to set or change ERRLOG operations) from a nonprivileged terminal. Nonprivileged users can only issue ELI Show commands.

User Action: Log on a privileged terminal and issue the commands.

ERRLOG -- Task subpacket corrupted

Explanation: ERRLOG tried to use a corrupted task subpacket.

User Action: Submit an SPR.

ERROR LOG TASK (ERRLOG) AND ERROR LOG INTERFACE (ELI)

ERRLOG -- Unable to open file

Explanation: ERRLOG could not open the log file to begin logging. ERRLOG then transfers logging to the backup file immediately.

User Action: Issue an ELI command to establish a new backup file.

ERRLOG -- Unknown command packet subtype

Explanation: ERRLOG encountered an unknown command packet subtype.

User Action: If the Error Logging System includes a user-written control file module to generate Error Log Packets, check the code. Otherwise, submit an SPR.

ERRLOG -- **WARNING: Device ddnn: exceeded xx Limit (x)

Explanation: Device ddnn exceeded the error limit set with an ELI Hard or Soft Limit switch or the default error limit of five hard errors and eight soft errors.

User Action: Check to see if the number of errors indicates a serious hardware malfunction. To continue logging on the device, reset the QIO and error counts to zero with the /Reset switch or change the limits using the /HL or /SL switch.

CHAPTER 3

REPORT GENERATOR TASK (RPT)

This chapter describes how to use the Report Generator Task (RPT) to create Error Log Reports.

Chapter 1 provided an overview of the interaction of elements in the Error Logging System (the Error Log Control File and the Control File Language Compiler). The RPT switches described in this chapter use modules from the Error Log Control File to determine how to interpret and format information from the error log file. (See Chapter 2 for a description of how the Error Logger creates the error log file.) RPT and modules in the error log control file work together to interpret the information in the error log file and define an event that occurs on a device. They do not analyze the event itself or attempt to diagnose hardware failures.

All RPT reports use the same entry number to refer the same Error Log Packet, so you can use RPT brief reports to isolate a device or specific events occurring on that device, and then specify entry numbers to generate a full report on only the specific events you want to look at in more detail. Note, however, that some ELI commands may change the packet number associated with an event. For example, appending a file to the error log file will change the packet numbers in the appended file.

3.1 INSTALLING AND RUNNING RPT

Since RPT is a nonprivileged task, any user can use it to create Error Log Reports when it is installed on the system. To install RPT, enter the following MCR command from a privileged terminal or as an entry in the system startup command file:

```
>INS $RPT(RET)
```

If RPT is not installed, you can invoke it from any terminal, using the following MCR command:

```
>RUN $RPT(RET)
RPT>
```

To invoke RPT when it is installed, issue the following MCR command from any terminal:

```
>RPT(RET)
RPT>
```

3.2 USING RPT TO CREATE ERROR LOG REPORTS

The Error Log Control file needs at least two types of information from RPT switches to generate Error Log Reports:

- How to select which Error Log Packets to analyze
- How to format the Error Log Packets

In addition, on RSX-11M-PLUS systems the control file needs a third kind of information.

- How to summarize the information from the Error Log Packets

Switches on the RPT command line provide this information, which is independent of the file specification they accompany.

3.2.1 The RPT Command Line

The only element you must specify in an RPT command line is the equals sign (=). All other file and switch specifications in the command line are optional.

The general format of an RPT command line is

```
[reportfile[/switches]]= [inputfile[/switches]]
```

reportfile

The name of the listing file that contains the Error Log Report.

Instead of a report file, you can specify TI: to send the report to your terminal. On RSX-11M-PLUS systems with transparent spooling you can specify LP: to send the report to the line printer.

switches

Optional switches to control how RPT selects, formats, and (on RSX-11M-PLUS) summarizes information from the error log file. You can use the same switches with either the report file specification or the input file specification on the command line. RPT uses the switches in the order you specify, but ignores which file specification they accompany.

input file

The only input file you can specify in the command line is the Error Log File, the disk file that the Error Logger creates.

RPT also uses a universal library of compiled control file modules as input. RPT looks first for the file LX:[1,6]ERRLOG.ULB. If it does not find it, RPT looks for the file LB:[1,6]ERRLOG.ULB. Use pseudo device LX: if you wish to save space on LB:. RPT includes this file by default and you cannot specify or change it from the command line, so it is not part of the format described above.

RPT can, however, prompt for the name of a universal library. If you want RPT to prompt you for the universal library name, you must edit the RPTBLD.BLD file and make the value of USERCM non-zero, then relink

REPORT GENERATOR TASK (RPT)

RPT. If you do make this alteration, note that it has the additional effect of preventing you from issuing an RPT command line from the MCR level. That is, the following is the way to invoke RPT.

```
>RPT
CTL> (universal library filespec)
RPT> command line
```

The RPT input and output files described above assume the defaults listed in Table 3-1, unless you specify otherwise in the command line.

Table 3-1
RPT File Specification Defaults

	Report File	Input File	Universal Library File ¹
Device:	SY0:	LB:	LX:, LB:
UIC:	Current UIC	[1,6]	[1,6]
File Name:	ERRREPORT	LOG	ERRLOG
File Type:	.LST	.ERR	.ULB
Version:	new	latest	latest

1. Not specified by user.

3.2.2 Using Multiple Qualifiers in RPT Command Lines

You can only specify each RPT switch once in a command line. However, some switches provide an alternative syntax that allows you to specify more than one argument for the switch.

To specify more than one argument for an RPT switch, use the following command syntax:

```
/switch:(qualifier1,qualifier2...qualifiern)
```

The parentheses, which are a required part of the command syntax, allow RPT to use more than one qualifier for the switch. If you do not specify the parenthesis, RPT displays the following message on your terminal and exits:

```
ERLCNF-F-SYNTAXERR command line syntax error
```

For example, to specify a report on more than one device, use the following RPT switch:

```
/DE:(DB,DM2:,DR)
```

RPT generates a report on all the DB and DR devices in your system, as well as device DM2:.

REPORT GENERATOR TASK (RPT)

The switches that permit you to specify multiple qualifiers in this way are:

- The DEVICE switch
- The PACKET switch
- The SERIAL switch (one drive and one pack serial number)
- The TYPE switch
- The SUMMARY switch (on RSX-11M-PLUS)

3.2.3 Using the Default RPT Command Line

To use the RPT default command line, enter the following command:

RPT>= (RET)

This command causes RPT to use the file specification defaults (listed in Table 3.1) and switch defaults (listed below). In general, this command creates a brief format report, without any summaries, using all of the Error Log Packets in the error log file.

The RPT default command line invokes the following switches:

/F[ORMAT]:B[RIEF]

Creates a brief format report containing one line for each error log packet described in the report. (See Section 3.3.2.)

/T[YPE]:A[LL]

Creates a report on packets describing all types of events: peripheral, processor, memory, control, and system information packets. (See Section 3.3.1.5.)

/DA[TE]:R[ANGE]:*:*

Creates a report on packets of all dates. (See Section 3.3.1.1.)

/DE[VICE]:ALL

Creates a report on all error logging devices in the system

/PA[CKET]:*:*

Creates a report for all packet numbers. (See Section 3.3.1.3.)

/SU[MMARY]:N[ONE]

Does not create summary reports of statistical information on all packets included in the report. This switch is available only on RSX-11M-PLUS systems. (See Section 3.3.3.)

/W[IDTH]:W[IDE]

Creates a wide (132 column) report.

3.3 RPT REPORT SWITCHES

This section describes the RPT switches, according to the RPT requirement that they fulfill. These switches tell RPT how to perform the following tasks:

- Select packets
- Format packets
- Summarize information from packets (on RSX-11M-PLUS only)

RPT syntax only requires that you specify enough characters in a command or qualifier to make it unique. For example, you can specify /T for the /TYPE switch, but you must specify /SU for the /SUMMARY switch to distinguish it from the /SERIAL switch.

The command line examples used throughout this chapter highlight the command they describe. Any switches not explained in the command descriptions assume the default values described in Section 3.2.3.

Table 3-2 summarizes the RPT report switches in alphabetical order.

Table 3-2
RPT Report Switches and Subswitches

Switch	Qualifiers	Function
/DA:qualifier (Date)	P[REVIOUS]:ndays R[ANGE]:start:end T[ODAY] Y[ESTERDAY]	Select packet based on date.
/DE:qualifier (Device)	device_name(,s) ALL	Select packets based on device.
/F:qualifier (Format)		Describes how RPT formats the error log packets.
	B[RIEF]	Display packets in brief format (one line for each packet).
	F[ULL]	Display all of the information in the specified packet.
	N[ONE]	Does not display information on a packet-by-packet basis.
	R[EGISTER]	Displays the same information as the FULL qualifier, but shows only the device registers on packets for peripheral errors.
/P:qualifier (Packet number)	nnnn.nnn nnnn.nnn[:mmmm.mmm]	Select packets based on packet number.

(continued on next page)

REPORT GENERATOR TASK (RPT)

Table 3-2 (Cont.)
RPT Report Switches and Subswitches

Switch	Qualifiers	Function
/R:qualifier (Report)	DAY MONTH SYSTEM WEEK user_string	Invokes a predefined string of switches for RPT to use. The qualifier can be one of four DIGITAL-defined strings or a user-defined switch string.
/SE:qualifier (Serial number)	D[RIVE]:number P[ACK]:number	Selects packets based on drive and/or pack serial number. The pack serial number is supplied only on MSCP and last-track devices.
/SU:qualifier (Summary)		Selects the type of summary report RPT generates (on RSX-11M-PLUS systems only).
	A[LL]	Selects all summary reports (History, Error, and Geometry).
	E[RROR]	Creates a summary report based only on device errors.
	G[EOMETRY]	Creates a summary report based on disk geometry (sector or track, for example).
	H[ISTORY]	Creates a summary report based on the error history of the device(s) specified.
	N[ONE]	Creates no summary report.
/T:qualifier (Type)		Selects packets based on packet type.
	A[LL]	Selects all packets in the Error Log File.
	C[ONTROL]	Selects command packets from the Error Log Interface (ELI).
	E[RRORS]	Selects packets from the processor, memory, and peripherals.
	M[EMORY]	Selects packets from events that occur in memory (such as memory parity errors).
	PE[RIPHERAL]	Selects packets from all peripheral devices that support Error Logging. This qualifier does not display system information (such as mounts or dismounts).

(continued on next page)

REPORT GENERATOR TASK (RPT)

Table 3-2 (Cont.)
RPT Report Switches and Subswitches

Switch	Qualifiers	Function
/T:qualifier (Type) (Cont.)	PR[OCESSOR]	Selects packets from events that occur in the CPU, such as unknown interrupts.
	S[YSTEM_INFO]	Selects packets from events that occur on the system but are not specifically tied to a single piece of hardware (such as time changes, system service messages, mounts and dismounts).
/V:volume_label (Volume label)		Selects packets based on volume label.
/W:qualifier (Width)	N[ARROW] W[IDE]	Selects the width of the report RPT creates (80 or 132 columns). The narrow width qualifier is ignored on summary reports.

3.3.1 Packet Selection Switches

The following switches tell RPT how to select which Error Log Packets to report on. This selection is based on an attribute of the device or the packet or on the date and time that the packet was created.

3.3.1.1 The Date Switch

```
/DA[TE]:qualifier

QUALIFIERS:

P[REVIOUS]:n days
R[ANGE]:start_date:end_date
T[ODAY]
Y[ESTERDAY]

DEFAULT:

/DA:R:***
```

The /DATE switch allows you to select packets based on the date that an event occurred. This switch includes qualifiers to specify a range of dates or to specify a particular day. DIGITAL also supplies switch strings to use with the /REPORT switch that use the /DATE switch to create reports for the previous week or month.

The RANGE qualifier accepts starting and ending dates in the standard RSX format:

```
DD-MMM-YY
(DD-MMM-YY HH:MM:SS)
```

REPORT GENERATOR TASK (RPT)

However, if you specify the second format, with time as well as date, the parentheses are a required part of the syntax.

When you use the starting date and ending date format, the starting date rounds off to a time of 00:00:00 and the ending date rounds off to 23:59:59.

The asterisk (*) used at the beginning of a range specification indicates any date through the specified ending date. For example, *:12-JAN-82 specifies all of the packets from the beginning of the error log file through January 12, 1982.

The asterisk (*) used at the end of a range specification indicates any date since the specified beginning date. For example, 4-FEB-82:* specifies all of the packets from 00:00:00 on February 4, 1982 through the end of the error log file.

3.3.1.2 The Device Switch

/DE[VICE]:qualifier

QUALIFIERS:

device_name(,s)

ALL

DEFAULT:

/DE:ALL

The /DEVICE switch allows you to select packets for a particular device, for more than one device, or for all the devices on the system. You can specify more than one device with the /DEVICE switch by using the special syntax described in Section 3.2.2.

RPT uses the following conventions for device names with the /DEVICE switch:

Mnemonic	Meaning
dd	Selects all devices with the mnemonic dd.
ddnn:	Selects the device with the mnemonic dd and the unit number nn.

For example, /DE:DM selects all DM devices, and /DE:(DM,DB2:) selects all DM devices and device DB2:.

3.3.1.3 The Packet Switch

P[ACKET]:nnnn.nn[:mmm.mm]

DEFAULT:

/P:*:*

The /PACKET switch allows you to select a packet or range of packets by specifying the packet identification numbers. You can determine the packet numbers you want to see by examining a brief report of all packets.

REPORT GENERATOR TASK (RPT)

To select just one packet you specify one packet number. For example, /PA:123.4 selects only packet number 123.4. To select a range of packets, you specify the first and last packet numbers of that range: /PA:123.4:432.1 selects all the packets from packet 123.4 through packet 432.1.

You can also specify more than one packet or packet range by using the special syntax described in Section 3.2.2.

The asterisk (*) indicates an open-ended number. You can select all the packets before a particular number (*:235.3), or all the packets after a particular number (235.3:*) .

3.3.1.4 The Drive and Pack Serial Number Switch

/SE[RIAL]:qualifier

QUALIFIERS:

D[RIVE]:serial_number
P[ACK]:serial_number
(D[RIVE]:serial_number,P[ACK]:serialnumber)
(P[ACK]:serial_number,D[RIVE]:serial number)

DEFAULT:

None

The /SERIAL switch allows you to select packets based on their drive or pack serial number or both. This switch only applies to peripheral errors. You can select packets from any device that has a serial number by drive serial number, but you can only select packets from MSCP and last track devices by pack serial number. Appendix B explains where RPT gets drive serial numbers and lists the significant digits in serial numbers for each error logging device.

You can specify one drive and one pack serial number or both in the same command line by using the special syntax described in Section 3.2.2.

3.3.1.5 The Type Switch

/T[YPE]:[qualifier]

QUALIFIERS:

A[LL]
C[ONTROL]
E[RRORS]
M[EMORY]
PE[RIPHERAL]
PR[OCESSOR]
S[YSTEM_INFORMATION]

DEFAULT:

/T:A

REPORT GENERATOR TASK (RPT)

The /TYPE switch selects Error Log Packets based on their packet type. You can select the following types of packets (or combination of types) with the appropriate /TYPE switch qualifier:

Qualifier	Packet Type
ALL	All Error Log Packets in the Error Log File.
CONTROL	Error Log Command Packets sent by the Error Log Interface (ELI).
ERRORS	All Error Log Packets from peripherals, processor, and memory.
MEMORY	Error Log Packets from events that occur in memory (such as memory parity errors).
PERIPHERAL	Error Log Packets from all peripheral devices that support Error Logging. This qualifier does not display system information (such as mounts and dismounts) for the devices. That information is displayed by the SYSTEM_INFO qualifier.
PROCESSOR	Error Log Packets from events that occur in the CPU, such as unknown interrupts.
SYSTEM_INFORMATION	Error Log Packets from events that occur on the system, but are not specifically tied to a single piece of hardware (such as time changes, system service messages mounts, and dismounts).

You can specify more than one type of packet by using the special syntax for the /TYPE switch, described in Section 3.2.2.

3.3.1.6 The Volume Label Switch

/V[OLUME]:volumelabel

DEFAULT:

None

The /VOLUME switch selects packets for peripheral errors based on the volume label.

For example:

=/T:PE/V:ERRLOGSYS

This command line specifies that RPT find the device or devices containing a volume with the label ERRLOGSYS and generate a report of peripheral errors on those devices. Since the /TYPE switch specification did not include system information, the report will not include mounts or dismounts for the devices.

3.3.2 Report Format Switch

```
/F[ORMAT]:qualifier
```

QUALIFIERS:

```
B[R]IEF
F[ULL]
R[EGISTER]
N[ONE]
```

DEFAULT:

```
/F:B
```

The /FORMAT switch tells RPT how to format a report from packets in the error log file. You can select reports in brief format (one line for each error), in full format (all the information from the error log packets specified) or in register format (dumping only the registers for device errors). The following sections describes qualifiers to the /FORMAT switch.

3.3.2.1 Brief Reports

```
/F[ORMAT]:B[R]IEF
```

Brief reports are short, one-line per packet, reports on selected packets.

The brief report shown in Example 3-1 displays one line of information about each of the error log packets specified in the RPT command line. The following list describes the sections in the brief report. The numbers in the list reflect the numbers of the sections in Example 3-1. Note that all these examples show wide width reports.

- ① The Error Log Packet Entry Number which describes the relative position in the Error Log File. This number does not change unless the file is changed, by an ELI/APPEND command, for example. It is not changed by normal logging into the file.
- ② The date and time the packet was logged.
- ③ The type of entry in the error log file; for example, hard or soft device errors or system information.
- ④ The device on which the error occurred.
- ⑤ The error type as defined by the hardware information. RPT does not do any interpretation of these errors; it merely reports the hardware information.
- ⑥ Any other information error logging gathers on the error, such as the I/O function that occurred at the time of the error.

The following sections only appear on RSX-11M-PLUS reports:

- ⑦ The RPT command line that generated the report.
- ⑧ Input and report file specifications.

REPORT GENERATOR TASK (RPT)

- ⑨ The format selection.
- ⑩ The packet time range selection (time the packets were created).
- ⑪ The volume label selection.
- ⑫ The drive and pack serial number selections.
- ⑬ Types of summary reports selected.
- ⑭ The packet type selections.
- ⑮ The packet number selections.
- ⑯ The device selections.
- ⑰ The number of packets printed and processed.
- ⑱ The time RPT report generation began and ended.

Example 3-1 Error Log Brief Report

RSX-11M/M-Plus Error Logging System		Version V0-1	11-MAR-1983 08:27:50		Page 1
Entry		Time Stamp	Entry Type	Device	Error Type
4.4	04-JAN-1983 09:51:00	Device Hard Error	DL000:	Cover Open	Function = Read Data
5.1	04-JAN-1983 09:51:06	Device Hard Error	DL000:	Data CRC Error	Function = Read Data
5.2	04-JAN-1983 09:51:07	Device Hard Error	DL000:	Data CRC Error	Function = Read Data
10.3	14-JAN-1983 14:20:18	Device Soft Error	DM001:	Data Check	Function = Read Data
42.2	15-FEB-1983 14:02:23	Device Soft Error	MS000:	Uncorrectable Data	Function = Write

Selection Information:

- 7 Command line: EXMBRIEF1.RPT=RASIN.LOG/PA:(4.4,5.1,5.2,6.4,10.3,42.2)/W:N
- 8 Report file: DB2:[303,12]EXMBRIEF1.RPT;3
Input file: DB2:[303,12]RASIN.LOG;1
- 9 Report format selection:
BRIEF
- 10 Packet time selections:
From * through *
- 11 Volume label selection:
(not used)
- 12 Serial number selections:
Drive: (not used)
Pack: (not used)
- 13 Summary selections:
No History
No Error
No Geometry

Example 3-1 (Cont.) Error Log Brief Report

Selection Information:

14 Packet type selections:

Processor
Memory
System_Info
Peripheral
Control

15 Packet number selections:

4.4
5.1
5.2
6.4
10.3
42.2

16 Device selections:

(ALL)

17 Number of packets printed / processed:

6. / 6.

18 Processing began at 11-MAR-1983 18:12:02
Processing ended at 11-MAR-1983 18:12:20

3.3.2.2 Full Reports

/F[ORMAT]:F[ULL]

Full reports provide a detailed listing of device events. They list and interpret all of the information collected in the Error Log Packets they describe.

The full report, shown in Example 3-2, displays the complete contents of error log packet number 4.4, a Cover Open Error described in the brief report in Example 3-1. The following list describes the sections of the full report. The numbers on the list reflect the numbered sections in Example 3-2.

- ① The same identification information listed in items 1-5 of the brief report description.
- ② System identification information including operating system and base level, CPU type and address mapping type.
- ③ Device identification information including the device name, device type, volume label, controller, unit number, pack and drive serial numbers, total I/O count on the device, and the number of hard and soft errors logged previous to this one.
- ④ I/O operation identification includes the terminal and UIC that initiated the operation, the task name, the beginning physical memory address of the I/O buffer, the length of the I/O request (in bytes), the maximum number of retries the device driver allows for an I/O operation, the number of retries remaining, and the actual I/O operation taking place.
- ⑤ Concurrent I/O activity occurring on other devices when this error occurred. This section, which only appears on RSX-11M-PLUS systems, is only present when concurrent I/O activity takes place; otherwise the section is suppressed.
- ⑥ I/O operation information includes the device I/O function and type of error as defined by the hardware.
- ⑦ The device error position information locates the error by cylinder, group, head, sector, and logical block number.
- ⑧ The device-supplied information includes a dump of the device registers according to name, contents, and interpretation of the bits in the registers. The * beside some bit interpretations means that the condition is likely to have contributed to the error. It is a sign that you may want to examine the condition.

The following RPT command line generated the full report in Example 3-2:

```
RPT>EXEMPF.RPT=RAISIN.LOG/PA:4.4/F:F (RET)
```

Example 3-2 Error Log Full Report

Page 1

RSX-11M/M-Plus Error Logging System Version V0-1 21-JAN-1982 06:54:13

① Entry 4.4 Sequence 1. DL000: Device Hard Error (Cover Open) 04-MAY-1981 09:51:00

System Identification:

② System Ident Processor Mapping CPU Format

RSX-11M-PLUS 10 PDP-11/70 22-Bit CPA 1.

Device Identification Information:

③ Device Type Volume Label Controller Unit Subunit Pack SN Drive SN I/O Count Hard Errors Soft Errors

DL000: RL01 <null label> DL A 0 N/A N/A N/A 292. 0. 0.

I/O Operation Identification:

④ TI: UIC Task Name Address Length Maximum Retries Retries Remaining Operation

TT000: [003,054] ...BAD 340000 10240. 8. 8. IO.RLB ! IQ.X

Concurrent I/O Activity:

⑤ Device Controller Unit Subunit TI: UIC Task Address Length Operation

DR000: DR L 0 N/A C0000: [031,076] DR00F1 1212360 512. IO.RLB

MM000: MM T 0 0 TT0003: [031,076] BRUT3 N/A N/A IO.SPF

Example 3-2 (Cont.) Error Log Full Report

I/O Operation Information:

⑥ Device Function	Type of Error
Read Data	Cover Open

Device Error Position Information:

⑦ Cylinder Group Head Sector Block
173. N/A 0 13. 6926.

Device Supplied Information:

⑧ Name	Value	Interpretation
RLCS	104335	* [15] Composite Error [9: 8] Drive Selected = 0 [6] Interrupt Enabled [3: 1] Function = Read Data
RLBA	043000	[15: 0] Bus Address Register
RLDA	126716	[15: 7] Cylinder Address = 173. [5: 0] Sector Address = 14.
RLMP1	133333	[12: 0] Word Count = 9685. words remaining
RLMP2	046074	* [14] Current Head Error * [10] Write Gate Error [6] Head Address = Upper head [4] Heads Out (over the disk) [2: 0] Drive State = Seek
		* [11] Data CRC Error [7] Controller Ready = 01 (B) [5: 4] BA17, BA16 [0] Drive Ready
		[6] Head Selected = Lower head
		* [11] Spin Speed Error [7] Drive Type = RL01 * [5] Cover Open [3] Brushes Home

REPORT GENERATOR TASK (RPT)

3.3.2.3 Register Reports

/F[ORMAT]:R[EGISTER]

Register reports contain the same information as full reports for all events except those that occur on peripherals. Register reports list the contents of all device registers for peripherals, but contain no other information.

The register report in Example 3-3 includes only the register section of the full report for packet 4.4 (the Cover Open Error).

The following RPT command line generated the Register Report shown in Example 3-3:

RPT>EXEMP.N.RPT=RAISIN.LOG/F:R/PA:4.4 (RET)

Example 3-3 Error Log Register Report

Page 1

RSX-11M/M-Plus Error Logging System Version V0-1 22-JAN-1982 08:31:15
 Entry 4.4 Sequence 1. DL000: Device Hard Error (Cover Open) 04-MAY-1981 09:51:00

Device Supplied Information:

Name	Value	Interpretation
RLCS	104335	*[15] Composite Error [9: 8] Drive Selected = 0 [6] Interrupt Enabled [3: 1] Function = Read Data *[11] Data CRC Error [7] Controller Ready = 01 (B) [5: 4] BA17,BA16 [0] Drive Ready
RLBA	043000	[15: 0] Bus Address Register
RLDA	126716	[15: 7] Cylinder Address = 173. [5: 0] Sector Address = 14.
RLMP1	133333	[12: 0] Word Count = 9685. words remaining
RLMP2	046074	*[14] Current Head Error *[10] Write Gate Error [6] Head Address = Upper head [4] Heads Out (over the disk) [2: 0] Drive State = Seek [6] Head Selected = Lower head *[11] Spin Speed Error [7] Drive Type = RL01 *[5] Cover Open [3] Brushes Home

3.3.2.4 No Report

```
/F[ORMAT]:N[ONE]
```

RPT does not generate a formatted output report on event information. This switch satisfies the requirement to tell RPT how to format the packets by telling it not to format the packets or produce a packet-by-packet report. It is useful on RSX-11M-PLUS systems when you only want to generate a summary report.

3.3.3 Summary Switch (RSX-11M-PLUS only)

```
/SU[MMARY]:qualifier
```

QUALIFIERS:

```
A[LL]
E[RROR]
G[EOMETRY]
H[ISTORY]
N[ONE]
```

DEFAULT:

```
/SU:N
```

The /SUMMARY switch, which is only available on RSX-11M-PLUS systems, tells RPT how to summarize the information from packets in the error log file. Since the summaries are compilations of the data gathered from the individual packets, the /SUMMARY switch tells RPT what particular piece of information from the packets to use as the basis for a summary report.

RPT cannot create summary reports in narrow width. If you specify narrow width, with the /W:N command, RPT formats the packet-by-packet display in narrow width, but formats the summary in wide width.

The following sections describe the summary reports you can generate with /SUMMARY qualifiers.

3.3.3.1 The All Qualifier

```
/SU[MMARY]:A[LL]
```

RPT generates summary reports sorted by history, error, and geometry. These summary reports are described in Sections 3.3.3.2 through 3.3.3.4.

3.3.3.2 The Error Qualifier

```
/SU[MMARY]:E[RROR]
```

RPT generates a summary report sorted by error type. The error summary, sorted by device, shows the number of times each error occurred on the device. The Count column of the summary tells the number of times the error occurred. Example 3-4 shows the summary section of an error summary report.

REPORT GENERATOR TASK (RPT)

The following RPT command generated the report in Example 3-4:

```
>RPT ERRORRPT.LOG=/SU:E/F:N (RET)
```

When you specify /FORMAT:NONE, RPT does not display packets on a packet-by-packet basis as shown in the previous examples.

Example 3-4 Error Summary Report

Error Summary (sorted by device):

Device	Type	Drive SN	Volume Label	Pack SN	Error Type	Count	First/Last Occurrence	Entry
DM000:	RK06	FFF	<null label>	N/A	Data Late	4.	01-DEC-1981 15:39:26	2.2
							01-DEC-1981 15:40:23	3.2
DS001:	RS03	N/A	WORKVOL	N/A	Nonexistent Drive	3.	01-DEC-1981 15:50:43	4.3
							01-DEC-1981 16:21:41	7.3
							01-DEC-1981 16:21:45	8.1
MM001:	TU45	148	<null label>	N/A	No error bit found	1.	01-DEC-1981 16:21:45	8.1
							01-DEC-1981 16:05:30	5.2
							01-DEC-1981 16:05:30	5.2
							01-DEC-1981 16:05:44	5.3
					LRC Error (NRZI)	4.	01-DEC-1981 16:08:15	7.1

REPORT GENERATOR TASK (RPT)

3.3.3.3 The Geometry Qualifier

/SU[MMARY]:G[EOMETRY]

RPT generates a summary report based on device geometry (logical block or sector, for example). The Error Count column of the summary tells how many times an error occurred in that device location.

The following RPT command generated the report in Example 3-5:

>RPT ERRORRPT.LOG=/SU:G/F:N (RET)

Example 3-5 Geometry Summary Report

Geometry Summary (sorted by device):

Device	Type	Drive SN	Volume Label	Pack SN	Head	Group	Cylinder	Sector	LBN	Error Count
DM000:	RK06	FFF	<null label>	N/A	0.	N/A	111.	15.	7341.	1.
					2.	N/A	108.	18.	7190.	1.
					2.	N/A	113.	21.	7523.	1.
					2.	N/A	82.	0.	5456.	1.
DS001:	RS03	N/A	WORKVOL	N/A	0.	N/A	N/A	35.	8.	3.
RS04		N/A	WORKVOL	N/A	0.	N/A	N/A	35.	17.	1.

REPORT GENERATOR TASK (RPT)

3.3.3.4 The History Qualifier

/SU[MMARY]:H[ISTORY]

RPT generates a summary report sorted by device error history. It displays the hard and soft error count and QIO count for every volume used on each device.

The following RPT command line generated the report in Example 3-6:

>RPT ERRORRPT.LOG=/SU:H/F:N (RET)

Example 3-6 History Summary Report

History Summary (sorted by device):

Device	Type	Volume Label	Pack SN	Total QIOs	Hard Errors	Soft Errors
DB002:	RP04/05	WRTVOL	N/A	135683.	5.	
DK001:	RK05	SGNBKUP	N/A	6.		
DL001:	RL02	<null label> PURPLE	N/A 4167D	1193. 5992.		
DM000:	RK06	<null label>	N/A	1323.		4.
DY000:	RX02	TRANS	N/A	46.		
MM001:	TE/U16/45/77	<null label>	N/A	2213.	4.	

3.3.3.5 The None Qualifier

```
/SU[MMARY]:N[ONE]
```

RPT does not generate a summary report. However, this qualifier satisfies the RPT requirement on RSX-11M-PLUS that the command line specify how to summarize the information from Error Log Packets.

3.3.4 The Report Switch

```
/R[REPORT]:defined report string
```

DEFAULT:

None

The /REPORT switch invokes a predefined string of switches for RPT to use. This switch string usually defines a particular type of report, such as a report for a particular time period. The string contains any legal combination of RPT switches. The string cannot include the /REPORT switch.

The /REPORT switch allows you to access a file that contains the switch combinations you use frequently and lets you invoke the switches, using the string name, instead of reentering the switches explicitly.

RPT uses the normal default values described in Section 3.2.1 for all switches not defined in the switch string if the switches have defaults.

The DIGITAL and user-defined switch strings are found in the Control File Module, PARSEM, or in a disk file, LB:[1,6]ERRDEFINE.CFS, respectively. The /REPORT switch first searches PARSEM, where it finds DIGITAL-defined strings. If the string is not defined there, RPT searches ERRDEFINE.CFS.

Since RPT looks in the Control File Module first, you cannot redefine the DIGITAL-supplied strings unless you alter the control file module. DIGITAL does not recommend that you alter control file modules. You can change the definitions for DIGITAL-supplied strings by slightly altering their names and inserting the switch under the new name in ERRDEFINE.CFS.

3.3.4.1 Predefined Switch Strings

DIGITAL supplies four predefined switch strings to use with the /REPORT switch.

On RSX-11M-PLUS systems, the switch strings define:

- DAY - /FO:FULL/SU:ALL/DA:TODAY
- WEEK - /SU:(HISTORY,ERROR)/DA:PREVIOUS:7
- MONTH - /SU:(HISTORY,ERROR)/DA:PREVIOUS:31
- SYSTEM - /SU:(HISTORY,ERROR)

On RSX-11M systems, the switch strings define:

- DAY - /FORMAT:FULL/DA:TODAY
- WEEK - /DA:PREVIOUS:7
- MONTH - /DA:PREVIOUS:31
- SYSTEM - uses all default switches

Note that the names of the predefined switch strings must be entered in full. They cannot be abbreviated.

3.3.4.2 User Defined Switch Strings

You can name and define your own switch strings to use with the /REPORT switch by creating and editing LB:[1,6]ERRDEFINE.CFS and inserting the switch strings you want to define.

Entries in this file must be in the form:

```
'switchname','switchstring'
```

Note that single quotation marks are a required part of the syntax.

switchname

The name of the switch string you are defining. This name becomes the qualifier to the /REPORT switch when you want to invoke the string. (The name must be nine characters or less.)

switchstring

The RPT switches you select to generate the report. (The switch string must be 80 characters or less.)

For example, if you want to generate a brief report of peripheral errors on all the DB devices on your system, edit ERRDEFINE.CFS and insert the following line:

```
'DB','/FO:B/TY:PE/DE:DB'
```

You can then create this report with the following RPT command:

```
RPT>outfile=infile/R:DB
```

When you invoke a user-defined string, you must enter the full switch string name.

3.3.5 The Width Switch

```
/W[IDTH]:qualifier
```

QUALIFIERS:

N[ARROW]

W[IDE]

DEFAULT:

```
/W:W
```


REPORT GENERATOR TASK (RPT)

The /WIDTH switch allows you to set the line width of the report RPT generates to narrow (80 columns) or wide (132 columns). The basic report format does not change when RPT creates a narrow report. Instead, each long line of the report wraps onto the next line at an appropriate place.

Note that Summary reports do not honor the /WIDTH switch. The summary portion of these reports is always WIDE format.

3.4 ERLCNF REPORT MESSAGES

The Error Log Control File displays messages on your terminal if errors occur during report generation. The messages include an abbreviation, a severity level code for the error (warning, informational, or fatal), and text describing the error.

In some cases, RPT also writes the message in the Error Log Report, if it explains an error that appears in the report. For example, when RPT fails to find a control file module for a device you specify, it displays a message on your terminal and in the report that includes the error message.

This section lists the ERLCNF messages, along with possible causes and methods for recovery.

The following are Fatal ERLCNF errors:

ERLCNF-F-ARGNOTUNQ, Argument specification <argument> is not unique

Explanation: You did not specify enough characters in a switch argument to make it unique. It can be confused with another argument.

User Action: Check the argument syntax and reenter the command.

ERLCNF-W-BADSUBPKT, Possible corruption in the <packetname> subpacket in item <item label>

Explanation: RPT found something in the subpacket that appeared to be abnormal. The file may be corrupted or it may be an internal error within RPT.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have. You can create a dump of the packet using the starting virtual block number of the packet: the nnn portion of the packet number nnn.m.

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-W-DUILLFORM, MSCP format code <code> is undefined

Explanation: This may be an internal error within RPT. It indicates a format code in the RA80 packet that is corrupted.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

REPORT GENERATOR TASK (RPT)

ERLCNF-F-ILLARGCOM, Illegal argument combination

Explanation: You specified an illegal combination of arguments with a switch.

User Action: Check the syntax and reenter the command.

ERLCNF-F-ILLFILSPC, Illegal file specification - <filename>

Explanation: You used an illegal file specification with an RPT report generating command.

User Action: Check the syntax and try the operation again.

ERLCNF-W-ILLPACCOD, Illegal code in packet <packetid>, Code = <xx>

Explanation: The major code for the indicated packet is beyond the range that RPT can handle.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-F-ILLPACRAN, Illegal packet range - LOW = <xx>, HIGH = <xx>

Explanation: The RPT Packet Selection switch requires arguments to be packet numbers in a specific format.

User Action: Determine the correct number for the packet you want to display, check the syntax and reenter the command.

ERLCNF-W-ILLPACSBC, Illegal subcode in packet <packetid>, Code = <xx>, Subcode = <xx>

Explanation: The subcode for the indicated packet is beyond the range that RPT can handle.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-F-ILLSWTARG, Illegal switch argument - <argument>

Explanation: RPT recognized the switch argument, but determined that the argument is incorrect in the context given.

User Action: Check the syntax and reenter the command.

REPORT GENERATOR TASK (RPT)

ERLCNF-F-INTERR001, Internal error detected at position number <n>

Explanation: This is an internal RPT error. It occurs with the PARSECLST and PARSECTION error messages.

User Action: You should never see this message. If you do, send in an SPR and the command line that generated the message and any other information you have.

ERLCNF-F-MODNOTFND, Module not found - <module>

Explanation: RPT searched ERRLOG.ULB for the module and did not find it.

User Action: You should never see this message. If you do, send in an SPR and the command line that generated the message. Be sure to include the name of the module that was missing.

ERLCNF-F-MULARGSPC, Argument <argument> specified multiple times

Explanation: You specified an RPT switch argument more than once.

User Action: Check the syntax and reenter the command.

ERLCNF-F-MULSWTSPC, Switch <switch> specified multiple times

Explanation: You entered the specified switch more than once on the same RPT command line. RPT only allows you to specify each switch once.

User Action: Check the syntax and reenter the command. Use the special syntax for multiple switch specifications described in Chapter 3 if the switch allows it.

ERLCNF-W-NODACSPRT, No IO_ACTIVITY support, packet = (packet)

Explanation: This is usually caused by enabling I/O activity support on RSX-11M systems without enabling the corresponding support in the error log control file.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-F-NODIDPACK, No Device_ID subpacket

Explanation: This is an internal error within RPT.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have.

REPORT GENERATOR TASK (RPT)

ERLCNF-W-NODRIVSZ, No drive of size <size> for mnemonic <ddnn>; using EUNKWN

Explanation: This may be an internal error within RPT.

User Action: You should never see this message if you have all DIGITAL hardware. If you have non-DIGITAL hardware, and you receive this message, it is caused by a disagreement between RPT's table of device sizes and the actual size of the device. See Section 4.5.3.4 for information on changing the table of device sizes.

ERLCNF-W-NODRIVTYP, No drive type <type> for mnemonic <dd>; using EUNKWN

Explanation: This may be an internal error within RPT. From the mnemonic, the drive appears to be a MASSBUS device. However, RPT does not recognize the device type as a MASSBUS device.

User Action: You should never see this message if you have only DIGITAL hardware. If you have non-DIGITAL hardware, the error is caused by disagreement between RPT's table of device sizes and the size of the actual device. See Section 4.5.3.4 for information on changing the table of device sizes.

ERLCNF-F-NOINPFILE. No input file specified

Explanation: RPT did not find an input file on the command line. This message occurs when you failed to specify an equals (=) sign in the command.

User Action: Check the syntax and reenter the command.

ERLCNF-W-NONOTES, No notes available for device <devicename>

Explanation: RPT includes a facility for displaying notes at the bottom of Full or Register reports. This internal error message indicates that a device which did not have an associated NOTES module.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-F-NOREMATCH, No predefined switch string for <string>

Explanation: RPT did not find the defined report string you used in a /R[REPORT] command, either in ERRDEFINE.CFS or among the DIGITAL-defined report strings. Remember to use the entire name of the DIGITAL or user-defined string.

User Action: Check the syntax and reenter the command.

REPORT GENERATOR TASK (RPT)

ERLCNF-F-OPNINPFIL, Failed to open the input file

Explanation: RPT could not open the input file specified. This message is accompanied by the FILERRCOD information message, that displays the FCS error code from the file.

User Action: Check the FCS error code and retry after correcting the indicated condition.

ERLCNF-F-OPNREPFIL, Failed to open the report file

Explanation: RPT could not open the report (output) file specified. This message is accompanied by the FILERRCOD information message, that displays the FCS error code from the report file.

User Action: Check the FCS error code and retry after correcting the indicated condition.

ERLCNF-F-OPNUSRFIL, Failed to open the user file

Explanation: RPT could not open the user file specified. This message is accompanied by the FILERRCOD information message, that displays the FCS error code from the file.

User Action: Check the FCS error code and retry after correcting the indicated condition.

ERLCNF-F-SWTNOTUNQ, Switch specification <switch> is not unique

Explanation: You did not specify enough characters of a switch to make it unique. It can be confused with another switch.

User Action: Check the switch syntax and reenter the command.

ERLCNF-F-SYNTAXERR, Command line syntax error

Explanation: Some element of the command line does not have the correct syntax.

User Action: Check the syntax and reenter the command.

ERLCNF-F-TOOFEWARG, Too few arguments in switch <switch name>

Explanation: You specified a switch that requires one or more arguments, without specifying enough arguments.

User Action: Check the syntax and reenter the command.

ERLCNF-F-UNKNWARG, Unknown argument - <argument>

Explanation: You specified an argument that is unknown to RPT.

User Action: Check the syntax and reenter the command.

REPORT GENERATOR TASK (RPT)

ERLCNF-W-UNKNWNDEV, Device mnemonic <dd> is unknown; using EUNKWN

Explanation: This may be an internal error within RPT.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-W-UNKNWNNOT, No note number <number> for device <devicename>

Explanation: RPT includes a facility for displaying notes at the bottom of reports. This internal error message indicates that a device tried to print a note that was not available.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-F-UNKNWSWT, Unknown switch - <switchname>

Explanation: You specified an unknown RPT switch.

User Action: Check the syntax and reenter the command.

The following are ERLCNF Warning messages:

ERLCNF-W-USEEUNKWN, Module <modulename> not found; using EUNKWN

Explanation: RPT was not able to find the module specified in the Error Logging Universal Library and went to the EUNKWN module instead. This causes a formatted dump of the device register to appear in the report. This message usually occurs if you tune your ULB and eliminate the module for a device you want to use.

User Action: Retune the ULB to include the missing module.

The following are ERLCNF Informational messages. They accompany other ERLCNF messages to give you additional information. They do not affect RPT operation.

ERLCNF-I-FILERRCOD, File error code = <errorcode>

Explanation: This message displays the FCS error code for a file. It accompanies messages on file access failures.

User Action: None is necessary. This is an informational message.

REPORT GENERATOR TASK (RPT)

ERLCNF-I-PARSECLST, PARSE.SECTION_LIST = <buf>

Explanation: This is an internal error within RPT. This message accompanies the INTERR001 message described above.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

ERLCNF-I-PARSECTION, PARSE.SECTION = <buf>

Explanation: This is an internal error within RPT. It accompanies the INTERR001 message described above.

User Action: You should never see this message. If you do, send in an SPR along with a dump of the packet that generated the message and any other information you have (See BADSUBPKT description).

If the message refers to a packet that you have altered, or a module that you wrote, correct the module, recompile and add it to the library.

3.5 ERLRPT REPORT MESSAGES

Most of the following error messages are either associated with errors in the control file module that RPT is interpreting, or internal RPT errors.

If the message refers to a control file module that you have altered, or a module that you wrote and added to the error logging system, correct the error, recompile the module, and add it to the library. The module in which the error occurred is specified in the first (or top) line of the execution stack dump produced by RPT. This information appears on the report file and on the terminal from which RPT is being run.

If the message refers to a DIGITAL-supplied module or is an internal RPT error, please submit an SPR and include a listing of the error log report file produced by RPT.

ERLRPT-F-ACCUDFVAR, Attempt to access undefined variable.

Explanation: A control file module attempted to access a variable which had not been defined.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-BADDIGIT, Invalid numeric digit in conversion.

Explanation: A numeric literal or the ASCII string argument for the %COD\$OCTAL, %COD\$DECIMAL, %COD\$HEX, %COD\$BCD, %COD\$BINARY, or %COD\$MACHINE function contained an illegal character for the specified radix or was null or blank.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-BITFLDSIZ, Bit or field too large in extraction operation.

Explanation: The bit or field in an extraction operation exceeded the size of the value on which the extraction was performed.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-BITTOOHIG, Bit number too large for specified storage unit.

Explanation: The bit number specified by the character string portion of a #BI, #WI, #LI, #QI, or #VI numeric literal was too large for the specified value size.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-CASENOMAT, CASE selection expression has no matching value.

Explanation: No match was found for the value of the selector expression in a CASE statement, and no ELSE clause was specified in the CASE statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-CONTROLFI, Could not open control file.

Explanation: The control file module could not be opened.

User Action: If using the default control file library, check to see that it is in either LX:[1,6] or LB:[1,6] and is not locked, and that you have read access to it. If using a user specified control file, check to see that it is not locked and that you have read access.

ERLRPT-F-COROUMIS, COROUTINE statement executed with no COROUTINE stack frame.

Explanation: A COROUTINE statement was executed without specifying a coroutine in the corresponding CALL statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-CRASH, Control file requested abort.

Explanation: The CRASH statement was executed by a control file module.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DATNOTEXI, Data declaration is longer than data.

Explanation: The amount of data specified in a PACKET or SUBPACKET declaration was larger than the amount of data in the PACKET or SUBPACKET. This condition may be due to an error in the control file module or an error in the error log packet being analyzed.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DECAGAIN, Group in declaration already declared. Redecclaration illegal.

Explanation: A DECLARE, PACKET, SUBPACKET, TABLE, or DYNAMIC_TABLE statement was executed with a group name that was already defined.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DECGRPCTX, Group in DECODE statement has no context.

Explanation: The group in the DECODE statement was a TABLE, DYNAMIC_TABLE, or PACKET or SUBPACKET with the REPEATED attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DECNOBIT, No BIT declaration corresponding to DECODE list item.

Explanation: The bit number specified for a data item in the DECODE statement had no corresponding BIT declaration for the data item in the specified group.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DECNOTEXT, No bit to text translation for DECODE list item.

Explanation: The BIT declaration corresponding to the bit number specified for a data item in the DECODE statement, had no print expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-DEFCASELS, No match for control expression in CASE conditional definition.

Explanation: No match was found for the value of the selector expression in a CASE conditional definition and no ELSE clause was specified.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DEFNOCONT, Attempt to access data in variable in group with null context.

Explanation: The control file module attempted to access a variable in a TABLE, DYNAMIC TABLE, or PACKET or SUBPACKET with the REPEATED attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-DEFNOSTAK, Declaration stack overflow.

Explanation: The stack used for processing declarations has overflowed.

User Action: Edit RPTBLD.CMD to increase the psect extension for psect DCSTK0, and rebuild RPT.

ERLRPT-F-DEFSTKUND, Internal error - Declaration stack underflow.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-DIVZERO, Attempt to divide by zero.

Explanation: A control file module attempted to divide by zero.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-EXEINVCOD, Internal error - Execution stack entry has invalid code.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-EXEINVPOS, Internal error - INPUT file has invalid position value.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-EXPGRPNOC, Attempt to reference POINTER for group without context.

Explanation: A control file module attempted to reference the POINTER special variable for a TABLE, DYNAMIC_TABLE, or PACKET or SUBPACKET with the REPEATED attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-EXPINVCOD, Internal error - Invalid expression item code in expression.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-EXPINVTYP, Internal error - invalid symbol data type in expression.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-EXPNORSYM, Symbol without read access referenced in expression.

Explanation: A control file module attempted to read a variable defined in a DECLARE statement, which had not been initialized.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-EXPUDFGRP, Undefined group referenced in expression.

Explanation: A control file module attempted to reference a group which had not been defined.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-EXPUDFSYM, Undefined symbol referenced in expression evaluation.

Explanation: A control file module attempted to access an undefined symbol.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-EXPVALOVR, Value stack overflow during expression evaluation.

Explanation: The stack used for processing values and expressions has overflowed.

User Action: Edit RPTBLD.CMD to increase the psect extension for psect VLSTK0, and rebuild RPT.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-EXPVARNOC, Attempt to access variable without context in expression.

Explanation: A control file module attempted to reference a variable in a TABLE, DYNAMIC TABLE, or PACKET or SUBPACKET with the REPEATED attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FILERCLOS, File close error.

Explanation: An error occurred when RPT attempted to close a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLRPT-F-FILERREAD, File read error.

Explanation: An error occurred when RPT attempted to read a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLRPT-F-FILERSPAN, Records in file are not allowed to span blocks.

Explanation: The span block attribute of the error log file being analyzed was set. ELI creates the error log file with this attribute set, and neither ELI, ERRLOG, nor RPT will modify it, but some other task may have.

User Action: Use ELI to (re)start error logging with a new version of the error log file, then use PIP to append the previous version to the new version. PIP may produce the following warning message:

PIP -- Input files have conflicting attributes

This message can be ignored.

ERLRPT-F-FILERWRIT, File write error.

Explanation: An error occurred when RPT attempted to write to a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLRPT-F-FILINTOPN, Internal error - file already open.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-FILINVCOD, Internal error - invalid file code for specified operation.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-FILINVMOD, Control file library has invalid module name table format.

Explanation: The control file library has an invalid module name table format. The control file must be a universal library.

User Action: Make sure that the control file is a valid universal library and rerun RPT.

ERLRPT-F-FILNOTCTX, Operation requires that dynamic file have context.

Explanation: A control file module executed a POINTER DELETE or POINTER MOVE statement on a DYNAMIC_TABLE for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FILNOTEXI, Internal error - declared dynamic file does not exist.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-FILNOTVIR, Could not create virtual address space for module table.

Explanation: RPT could not dynamically extend its address space to create room for the module table.

User Action: If the maximum task size for the partition is less than 32K, use the MCR command SET /MAXEXT or DCL command SET SYSTEM/EXTENSION LIMIT to increase the maximum task size, or run RPT in a different partition.

ERLRPT-F-FILTOOBIG, File too large to read.

Explanation: RPT cannot analyze error log files which are larger than 65535 blocks.

User Action: Use ELI to create new error log files more often.

ERLRPT-F-FINDFIELD, FIELD in FIND statement does not have valid data type.

Explanation: A control file module executed a FIND statement where the specified FIELD was not NUMERIC, STRING, ASCII, RSXTIME, VMSTIME, or LOGICAL.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-FINDNOCON, FIND statement not valid on a group with no context.

Explanation: A control file module executed a FIND statement for a TABLE or DYNAMIC TABLE attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORCLSNUL, FORMAT clause null.

Explanation: A control file module executed a WRITE or WRITE_GROUP statement with a null FORMAT clause.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORFIELDS, FORMAT error - Field too narrow for variable to print.

Explanation: A control file module executed a WRITE_GROUP statement where the width specified by a !DP directive was too short for the corresponding variable.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORFIELDW, FORMAT error - Name too long for field in !DF directive.

Explanation: A control file module executed a WRITE_GROUP statement where the width specified in a !DF directive was less than the length of the name of the corresponding variable

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORINVCHA, FORMAT error - Invalid character in FORMAT clause.

Explanation: A control file module executed a WRITE or WRITE_GROUP statement with a FORMAT clause containing a non-printing character.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORINVDIR, FORMAT error - Invalid format directive code.

Explanation: A control file module executed a WRITE or WRITE_GROUP statement with a FORMAT clause containing an invalid format directive.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-FORINVVTY, FORMAT error - Attempt to output invalid variable type.

Explanation: A control file module executed a WRITE or WRITE GROUP statement with a FORMAT clause containing a !DP directive for which the corresponding variable was the wrong type.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORLINEOV, FORMAT error - Line overflow in FORMAT clause.

Explanation: A control module executed a WRITE or WRITE GROUP statement during which the output buffer overflowed while processing the FORMAT clause. The output buffer is 132 characters wide.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORNOARG, FORMAT error - Format directive missing required argument.

Explanation: A control file module executed a WRITE or WRITE GROUP statement with a FORMAT clause containing an !FC or !FS directive with no numeric argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORNONAME, FORMAT error - request to print a field name for a value.

Explanation: A control file module executed a WRITE or WRITE GROUP statement with a FORMAT clause containing a !DF directive matched with a value rather than a variable.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORNOREAD, FORMAT error - Attempt to print a variable without read access.

Explanation: A control file module executed a WRITE or WRITE GROUP statement which attempted to print a variable without read access.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FORNOTASC, FORMAT clause not ASCII.

Explanation: A control file module executed a WRITE or WRITE GROUP statement with a non-ASCII FORMAT clause.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-FUNDATNOT, Specified (sub)packet is not large enough for offset.

Explanation: A control file module executed a look-ahead function where the value of the offset argument was larger than the specified PACKET or SUBPACKET.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FUNFIELDS, Invalid conversion code argument to time conversion function.

Explanation: A control file module executed a time conversion function with an illegal value for the conversion code argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FUNINVPOI, Invalid string pointer value in string function.

Explanation: A control file module executed a %STR\$PARSE or %STR\$QUOTE function where the value of the pointer argument was larger than the length of the string argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FUNNOTCHA, Argument to STR\$CHAR is not in valid range for character.

Explanation: The value of the argument for the %STR\$CHAR function must be in the range 0 to 127(10).

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FUNNOTIMP, Function not implemented.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-FUNQUOODD, Quote string in STR\$QUOTE function must have even length.

Explanation: A control file module executed a %STR\$QUOTE function, where the quote string argument was not an even length.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-FUNSTRSIZ, Output string from string function too large.

Explanation: A control file module executed a string function which resulted in a string longer than 255 characters.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-FUNWRONGA, Incorrect number of arguments in function call.

Explanation: A control file module executed a function call with the wrong number of arguments.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-GROUPDEF, Attempt to reference undefined group.

Explanation: A control file module attempted to reference an undefined group.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-GROUPNOC, POINTER statement executed on a group without context.

Explanation: A control file module executed a POINTER statement on TABLE or DYNAMIC_TABLE for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-HEAPOVERF, Heap too small to hold value. Overflow.

Explanation: The heap used for processing values and expressions has overflowed.

User Action: Edit RPTBLD.CMD to increase the psect extension for psect VHEAP0, and rebuild RPT.

ERLRPT-F-INCFORWRI, Too few FORMAT expressions in WRITE_GROUP statement.

Explanation: A control file module executed a WRITE_GROUP statement which did not have two FORMAT expressions in the FORMAT clause.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-INCRDECRL, Numeric variable in INCREMENT or DECREMENT larger than value.

Explanation: A control file module executed an INCREMENT or DECREMENT statement on a variable which was larger than a word.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-INCRDECRN, Variable in INCREMENT or DECREMENT statement not numeric.

Explanation: A control file module executed an INCREMENT or DECREMENT statement on a non-numeric variable.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-INCRDECRV, Variable in INCREMENT or DECREMENT not valid or read-only.

Explanation: A control file module executed an INCREMENT or DECREMENT statement on a variable which was not both readable and writeable.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-INTINVDEC, Internal error - Invalid declaration entry type in WRITEGROUP.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-INTVALSTK, Internal error - statement left information on value stack.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-INVRADCNV, Internal error - Invalid radix code for conversion.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-LEAVENOC, LEAVE statement executed outside of a conditional block.

Explanation: A control file module executed a LEAVE statement, which was not inside a loop statement block.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-LISTNOEXP, No expression in LIST for corresponding SEARCH variable.

Explanation: A control file module executed a SEARCH statement in which a match was found, but there were not enough expressions in the list element for the number of variables specified in the GET clause of the SEARCH statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-LISTNOMAT, Too many expressions in SEARCH statement for referenced LIST.

Explanation: A control file module executed a SEARCH statement in which there were too many search expressions for the specified LIST.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-LISTNOTDF, Group referenced in SEARCH statement is not defined.

Explanation: A control file module executed a SEARCH statement in which the name specified for the LIST was not defined.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-LISTNOTLS, Group referenced in SEARCH statement is not a LIST.

Explanation: A control file module executed a SEARCH statement in which the name specified for the LIST was not defined as a list.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MATDIFTYP, Values of differing type cannot be matched.

Explanation: A control file module executed a MATCH statement which tried to match values of differing types.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MATVALSIZ, Values of different size cannot be matched.

Explanation: A control file module executed a MATCH statement which tried to match values of differing size.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MEMALLFAI, Memory allocation failure - insufficient virtual memory.

Explanation: RPT could not dynamically extend its address space to create room for DYNAMIC_TABLES or control file modules.

User Action: If the maximum task size for the partition is less than 32K, use the MCR command SET /MAXEXT or DCL command SET SYSTEM/EXTENSION LIMIT to increase the maximum task size, or run RPT in a different partition. If this occurs while generating summaries for large numbers of packets, try reducing the amount of data needed by using RPT switches to reduce the number of packets analyzed for each summary.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-MEMINIFAI, Memory allocation initialization failure.

Explanation: RPT could not dynamically extend its address space to create room for its data structures.

User Action: If the maximum task size for the partition is less than 32K, use the MCR command SET /MAXEXT or DCL command SET SYSTEM/EXTENSION_LIMIT to increase the maximum task size, or run RPT in a different partition.

ERLRPT-F-MODLOAGRP, Undefined group referenced by module to be loaded.

Explanation: The control file module being loaded, attempted to reference an undefined group.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MODLOASYM, Undefined symbol in module to be loaded.

Explanation: The control file module being loaded, attempted to reference an undefined symbol.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MODNAMENL, Module name cannot be null.

Explanation: A control file module attempted to access another control file module which had a null or blank name.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-MODNOMEM, Insufficient free memory to load module.

Explanation: RPT could not dynamically extend its address space to create room for control file modules.

User Action: If the maximum task size for the partition is less than 32K, use the MCR command SET /MAXEXT or DCL command SET SYSTEM/EXTENSION_LIMIT to increase the maximum task size, or run RPT in a different partition. If this occurs while generating summaries for large numbers of packets, try reducing the amount of data needed by using other switches to reduce the number of packets analyzed for each summary.

ERLRPT-F-MODSTART, Starting module for execution not found.

Explanation: The control file library must contain a module named DISPATCH.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-MODZERO, Attempt to modulus by zero.

Explanation: A control file module attempted to perform a MOD by zero.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-NOMORESTK, Execution stack overflow.

Explanation: RPT's execution stack has overflowed.

User Action: Edit RPTBLD.CMD to increase the psect extension for psect XCSTK0, and rebuild RPT.

ERLRPT-F-NOSTACKE, Internal error - Pop from execution stack with empty stack.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-NOTDYNFIL, Dynamic file operation performed on invalid group.

Explanation: A control file module specified a group which was not defined as DYNAMIC_TABLE in a statement or operation requiring a DYNAMIC_TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-NOTPOINT, POINTER LOAD or MOVE executed with a non-pointer variable.

Explanation: A control file module executed a POINTER LOAD or MOVE with a variable which was not a pointer.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-NOTPOIVAR, POINTER LOAD with no pointer variable specified.

Explanation: A control file module executed a POINTER LOAD or MOVE with no variable specified.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-NUMINVOPR, Invalid numeric double-operand operation code.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-OPRINVLOG, Attempt to perform logical operation on an invalid type.

Explanation: A control file module attempted to perform a logical operation with operands that were neither NUMERIC nor LOGICAL.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-OPRNOTIMP, Operation not implemented.

Explanation: A control file module attempted to perform a multiplication where both operands were larger than a word value.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-PACKETSIZ, Illegal packet size.

Explanation: The size of an error log packet was zero or would cause the packet to cross a block boundary.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-POISETGRP, POINTER variable is not from correct group in POINTER ... LOAD or MOVE.

Explanation: A control file module executed a POINTER LOAD or MOVE statement in which the optional pointer variable was not a pointer to the specified DYNAMIC_TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-POISETMOD, POINTER variable is from wrong module in POINTER ... LOAD or MOVE.

Explanation: A control file module executed a POINTER LOAD or MOVE statement in which the DYNAMIC_TABLE pointed to by the optional pointer variable was not in the same module as the DYNAMIC_TABLE specified in the POINTER statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-POISETSIZ, GROUP too small for POINTER in POINTER ... LOAD or MOVE.

Explanation: A control file module executed a POINTER LOAD or MOVE statement in which the optional pointer variable was pointing past the end of the specified DYNAMIC_TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-PROCNAME, Null procedure name.

Explanation: A control file module specified a null or blank procedure name in a CALL or ENABLE statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-RAD50BYTE, Cannot convert a byte using RAD50 conversion.

Explanation: A control file module attempted to convert an ASCII string or numeric literal to a BYTE using RAD50 conversion.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-RELINVCOD, Invalid relational operator.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-RETURNNO, A RETURN was executed with no corresponding CALL.

Explanation: A control file module executed a RETURN statement outside of a procedure or coroutine.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-SELECTNOM, SELECT statement index has no matching statement block.

Explanation: A control file module executed a SELECT statement with no statement block to match the value of the numeric control expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-SIGNOTASC, Parameter or message in SIGNAL-class statement not ASCII.

Explanation: A control file module executed a SIGNAL, SIGNAL_STOP, or MESSAGE statement with a non-ASCII argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-SIGTOOBIG, Message and parameters in SIGNAL-class statement too long.

Explanation: A control file module executed a SIGNAL, SIGNAL_STOP, or MESSAGE statement in which the length of the concatenated message and parameters was longer than 255 characters.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-SIGTOOMAN, Cannot issue a SIGNAL during SIGNAL processing.

Explanation: A control file module executed a SIGNAL or SIGNAL_STOP statement while processing a previous SIGNAL or SIGNAL_STOP.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-STANOTIMP, Statement not implemented.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-STANOTVAL, Internal error - invalid statement code.

Explanation: This is an internal error within RPT.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-SUBEXTBIG, Substring extraction end element exceeds string.

Explanation: A control file module attempted to perform a substring extraction in which the substring exceeded the end of the string.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-SUBPKTSIZ, Illegal subpacket size.

Explanation: The current subpacket, exceeded the bounds of the packet.

User Action: Please submit an SPR with any information you have.

ERLRPT-F-UNDEFPROC, Specified procedure not found.

Explanation: A control file module has executed a CALL statement, and the specified procedure was not found.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-UNDMODULE, Specified module not found.

Explanation: A control file module has executed a CALL statement, and the specified module was not found.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-VALSTKOV, Value stack overflow.

Explanation: The stack used for processing values and expressions has overflowed.

User Action: Edit RPTBLD.CMD to increase the psect extension for psect VLSTK0, and rebuild RPT.

ERLRPT-F-VALUESIZE, Value in expression is too large.

Explanation: A control file module evaluated an expression in which an intermediate value or the final value was too large.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-VALUETYPE, Value in expression is wrong type.

Explanation: A control file module evaluated an expression in which an intermediate value or the final value was the wrong type.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-VARNOCONT, Attempt to access variable in group without context.

Explanation: A control file module attempted to reference a variable for a TABLE, DYNAMIC TABLE, or PACKET or SUBPACKET with the REPEATED attribute for which the current record context was not valid.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-VARNODATA, Attempt to access variable in group with no data.

Explanation: A control file module attempted to reference a variable for a TABLE, DYNAMIC TABLE, or PACKET or SUBPACKET with no data.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLRPT-F-VARNOTDAT, Attempt to load data into a BIT or FIELD variable.

Explanation: A control file module attempted to load a value into a BIT or FIELD in a group, rather than into the data item for which the BIT or FIELD was defined.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

REPORT GENERATOR TASK (RPT)

ERLRPT-F-WRITEACCV, Attempt to load a value into a non-writable variable.

Explanation: A control file module attempted to load a value into a data item in a PACKET, SUBPACKET or TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CHAPTER 4

ERROR LOG CONTROL FILE ARCHITECTURE

This chapter describes the architecture of Error Log Control Files. A knowledgeable system programmer can use the information presented here to add user-written modules to the Error Logging System.

This chapter includes the following major sections:

- Terms and Concepts -- Defines the most important terms and concepts presented in this chapter.
- Control File Module Architecture -- Describes RSX-11M and RSX-11M-PLUS control file modules, the flow of program control through the modules, module compilation paths, and recompiling modules after modifications.
- Internal Interfaces -- Describes interaction between control file modules, with examples.
- Module Dispatching -- Explains event-level and device- or CPU-level dispatching.
- Support of Non-DIGITAL Devices -- Provides the information you need to include driver and error logging support for non-DIGITAL devices.
- Error Logging System Source Code Examples -- Includes annotated listings of source code for four modules: ERM23, DSP2M1, DSP2P1, and NRM23. The source code is keyed to discussions in the Internal Interfaces section (4.3) and Support of Non-DIGITAL Devices section (4.5) of this chapter.

4.1 TERMS AND CONCEPTS

Here are definitions of the most important terms and concepts presented in this chapter:

- Control File - A collection of modules that together perform a function, such as processing error log files.
- Module - A component of the Error Logging System. There are three kinds of modules: **source** modules, which have the file type .CNF, **object** modules, which have the file type .ICF, and **listing** modules, which have the file type .LST. Module names that end in M1 are generally common to both RSX-11M and RSX-11M-PLUS systems (for example, DEVSM1), except where the module name has an alternate P1 ending (for example, DSP2M1 and DSP2P1). In this case, module names that end in M1 are for RSX-11M systems only and those that end in P1 are for RSX-11M-PLUS systems only.

ERROR LOG CONTROL FILE ARCHITECTURE

- Control File Language - The language in which control files are written. The Control File Language (CFL) is described in Chapter 5.
- Error Log File - The file that contains the raw error logging data. One record in the file corresponds to one event. The default specification for this file is LB:[1,6]LOG.ERR.
- Event - Something that is logged in the error log file. An event may be the recording of an actual device error or it could be some informational data, such as a device mount or a change in system time.
- Packets/Subpackets - Each record (or event) is also a packet. A packet begins with a length word and is followed by data, which can consist of zero or more subpackets. A subpacket also consists of a length word followed by data. Every packet in the Error Logging System contains at least one subpacket.

NOTE

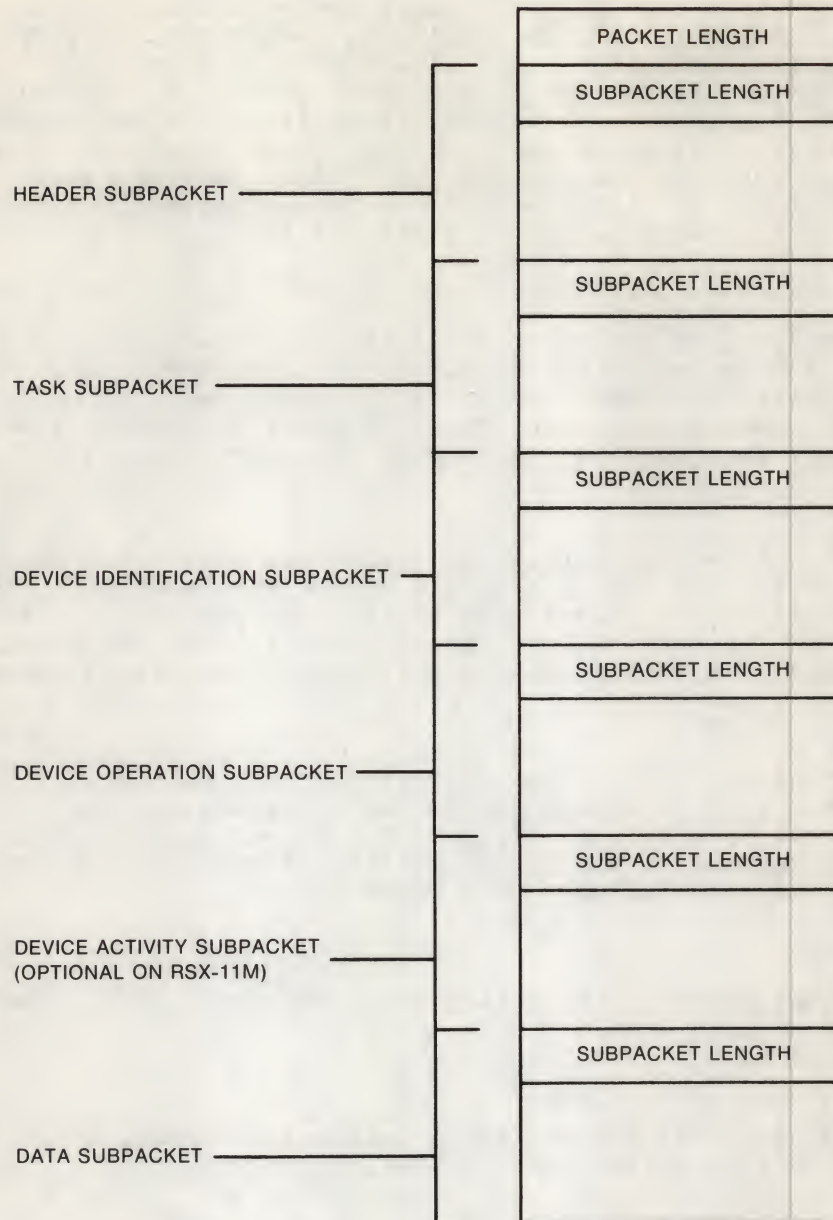
The packet length word begins the packet, but it is not part of the packet; the packet length word is kept by FCS. Therefore, the packet length word is not included in the length of the packet. However, the subpacket length words are part of the packet and are included in its length. This is consistent throughout the Error Logging System.

See Figure 4-1 for the general structure of an error-logging packet.

4.2 CONTROL FILE MODULE ARCHITECTURE

The Error Logging System is modular; that is, information and dependencies specific to different devices are isolated in modules written for each device. This section describes the architecture of the RSX-11M and RSX-11M-PLUS control file modules: the modules themselves, the flow of program control through the modules, the compilation paths, and modifying and recompiling the modules.

ERROR LOG CONTROL FILE ARCHITECTURE



ZK-1111-82

Figure 4-1 - Structure of Error-Logging Packet

4.2.1 RSX-11M and RSX-11M-PLUS Control File Modules

Here are short descriptions of the function of each of the RSX-11M and RSX-11M-PLUS control file modules. Remember that modules with names that end with "M1" are either common to both RSX-11M and RSX-11M-PLUS or are for RSX-11M only, and modules with names ending with "P1" are for RSX-11M-PLUS only.

DISPATCH

DISPATCH is the root module for the Error Logging System. It declares all commonly used variables, calls the INITM1 module to initialize the system, and then calls the PARSEM module to obtain and parse the command line. DISPATCH then requests the records

ERROR LOG CONTROL FILE ARCHITECTURE

from the input log file, declares the common subpackets (HEADER, TASK, DEVICE ID, DEVICE OPERATION, and DEVICE ACTIVITY (optional on RSX-11M, standard on RSX-11M-PLUS)) for each record, computes the correct dispatcher module name, and calls that module. When all the records are processed, it calls the summary modules if requested (RSX-11M-PLUS only), and finally calls the FINLM1/FINLP1 module to clean up. See Section 4.6.5 for the definitions of the standard DIGITAL subpackets. Dispatching is described in more detail in Section 4.4.

PARSEM

PARSEM declares variables local to the processing of the command line and calls the PARS1M module to obtain the command line. It then calls the PARS2M module to process any switches and the PARS3M module to open the various files. PARSEM also provides commonly used parsing routines to the other parsing modules.

PARS1M

PARS1M initializes parsing variables and gets the command line from RPT. It then breaks all of the file specifications out of the command line, leaving all of the switches. PARS1M then searches for the /REPORT switch. If it finds the switch, PARS1M replaces it with the specified string of predefined switches.

PARS2M

PARS2M gets a switch from the string of switches produced by PARS1M. It then checks the switch for ambiguity and calls PRS2AM to process the switch. If PRS2AM does not recognize the switch, it is passed to the PRS2BM module. PARS2M repeats this process until all switches have been processed.

PRS2AM

PRS2AM processes the following switches: DATE, DEVICE, and PACKET.

PRS2BM

PRS2BM processes the following switches: FORMAT, SERIAL, SUMMARY (RSX-11M-PLUS only), TYPE, VOLUME, and WIDTH.

PARS3M

PARS3M applies the default values to any switches that were not specified and opens the specified files.

SELTM1

SELTM1 is called by DISPATCH to determine if the current packet meets the selection criteria of the command line switches.

DSP1M1/DSP1P1

The DSP1M1/DSP1P1 modules process Error Log Control events (See Section 4.4.1). These modules declare the DATA subpacket for each type of event and process the event to completion, calling the formatter modules to print the common data if the FULL report format is specified.

ERROR LOG CONTROL FILE ARCHITECTURE

DSP2M1/DSP2P1

The DSP2M1/DSP2P1 modules process Device Error events (See Section 4.4.1). These modules call DEVSM1 to determine the name of the device-level module required to process the event and then calls that module as a co-routine and passes control to it. The device-level module declares the DATA subpacket and then extracts information from the registers of the logged device so it can provide additional selection information. When the device-level module returns control to DSP2M1/DSP2P1, it performs the last of the selection tests and makes the decision whether to continue with this event or not. If DSP2M1/DSP2P1 decides to continue, and if the FULL report format has been specified, DSP2M1/DSP2P1 calls the formatter modules to print the common information. Once printing is completed, control returns to the device-level module, which prints the device registers.

If the BRIEF report has been specified, DSP2M1/DSP2P1 still must decide whether to continue, but there is no need for the formatter modules and DSP2M1/DSP2P1 does its own printing.

On RSX-11M-PLUS only, once all the printing is completed, the Error Logging System tests to see if this entry belongs in any summaries. If so, DSP2P1 places the data relevant to each requested summary in the summary files.

DSP3M1/DSP3P1

The DSP3M1/DSP3P1 modules process Device Information events (See Section 4.4.1). They perform the same function as the DSP2M1/DSP2P1 modules, but for device errors not related to I/O. These modules are required only if you have a TU78 or MSCP (Mass Storage Control Protocol) device.

DSP4M1/DSP4P1

The DSP4M1/DSP4P1 modules process Device Control Information events (See Section 4.4.1). DSP4M1/DSP4P1 calls DEVSM1 to get the type of device associated with the device mnemonic.

Mount, dismount and reset operations have no DATA subpacket. The formatter modules print the information if the FULL report mode is specified; otherwise, the module does all the printing itself. Like the DSP2P1 and DSP3P1 modules, DSP4P1 records summary information if requested.

The Block Replacement event does have a DATA subpacket which is processed entirely by this module. This type of event does not contribute to summaries.

DSP5M1/DSP5P1

The DSP5M1/DSP5P1 modules process events detected by the CPU (See Section 4.4.3). DSP5M1/DSP5P1 gets the CPU type from the HEADER subpacket declared by DISPATCH and calls the appropriate CPU-level module as a co-routine if the event was a memory parity error. The processing then proceeds much like that for device errors.

If the event was an unknown interrupt, the module declares and processes the DATA subpacket itself.

ERROR LOG CONTROL FILE ARCHITECTURE

DSP6M1/DSP6P1

The DSP6M1/DSP6P1 modules process System Control Information events (See Section 4.4). There is no DATA subpacket associated with the power recovery event. The formatter modules print the common information if in FULL-report mode; otherwise, the module does all the printing itself.

DSP7M1/DSP7P1

The DSP7M1/DSP7P1 modules process Control Information events (See Section 4.4). These modules declare the DATA subpacket for each type of event and process the event to completion, calling the formatter modules to print the common data if the FULL report format is specified.

FINLM1/FINLP1

FINLM1 or FINLP1 is called by DISPATCH to clean up after all the error log events are processed.

On RSX-11M-PLUS only, FINLP1 also outputs the final page of the error log report. This page contains such information as the command line entered by the user, the files used, the switch states, the number of events processed, and how long it took to generate the report.

FM1NM1/FM1WM1

FM1NM1/FM1WM1 are formatter modules. They print information at the top of each page of a FULL report. The information comes mostly from the HEADER subpacket. FM1NM1 prints reports in NARROW format and FM1WM1 prints reports in WIDE format.

FM2CM1

FM2CM1 is one of the formatter modules. It prints the Requesting Task section of a FULL report. The information comes from the TASK subpacket. FM2CM1 prints reports in both NARROW and WIDE formats.

FM3CM1

FM3CM1 is a formatter module. It prints the Device Identification Information section of a FULL report. The information comes from the DEVICE ID subpacket. FM3CM1 prints reports in both NARROW and WIDE formats.

FM4NM1/FM4WM1

FM4NM1/FM4WM1 are formatter modules. They print the I/O Operation Identification section of a FULL report. The information comes from the DEVICE OPERATION subpacket. FM4NM1 prints reports in NARROW format, and FM4WM1 prints reports in WIDE format.

Optionally, FM4NM1/FM4WM1 also prints the Concurrent I/O Activity section of a FULL report. The information comes from the DEVICE ACTIVITY subpacket. See Section 4.1 for more information.

ERROR LOG CONTROL FILE ARCHITECTURE

FMTNP1/FMTWP1

FMTNP1/FMTWP1 are formatter modules. They print the first page of a FULL report, that is, all of the information from the HEADER, TASK, DEVICE ID, DEVICE OPERATION, and DEVICE ACTIVITY subpackets. FMTNP1 prints reports in NARROW format, and FMTWP1 prints reports in WIDE format.

INITM1

INITM1 initializes variables to be used later in the control file. It sets up the page-top banners, formatter module selectors, and WRITE_GROUP format statements, based on whether the report is NARROW or WIDE.

DEVSM1

DEVSM1 is called by DSP2M1/DSP2P1, DSP3M1/DSP3P1, and DSP4M1/DSP4P1 to provide certain device-related information. DSP2M1/DSP2P1 and DSP3M1/DSP3P1 call it to find, among other things, the name of the device-level module that should help process the event. DSP4M1/DSP4P1 calls DEVSM1 to find out the name of the device associated with a device mnemonic.

If the device mnemonic is DU, DEVSM1 then calls DEVUDA to do most of the processing.

DEVUDA

DEVUDA is called only by DEVSM1. It assists DEVSM1 in the processing of events on MSCP devices.

ERRORM

ERRORM is the error processor for the Error Logging System. Whenever a SIGNAL or SIGNAL_STOP occurs, ERRORM processes the error.

SMRYEP

SMRYEP prints Error summaries on RSX-11M-PLUS only. DISPATCH calls SMRYEP after all packets have been processed if an Error summary was requested.

SMRYGP

SMRYGP prints Geometry summaries on RSX-11M-PLUS only. DISPATCH calls SMRYGP after all packets have been processed if a Geometry summary was requested.

SMRYHP

SMRYHP prints History summaries on RSX-11M-PLUS only. DISPATCH calls SMRYHP after all packets have been processed if a History summary was requested.

CPU-level modules

There are five CPU-level modules, all with names derived from their associated processors. They are called as co-routines by DSP5M1/DSP5P1 to process memory parity errors.

- E1134 (RSX-11M only) - Processes errors from the PDP-11/34.

ERROR LOG CONTROL FILE ARCHITECTURE

- E1144 - Processes errors from the PDP-11/44.
- E1160 (RSX-11M only) - Processes errors from the PDP-11/60.
- E117X - Processes errors from the PDP-11/70 and PDP-11/74.
- E11XX - Processes errors from all other PDP-11 processors.

EUNKWN

EUNKWN is a universal device-level or CPU-level module. EUNKWN is called if a particular device-level module is unavailable, or if the device mnemonic is unknown to the Error Logging System. EUNKWN is also called if the CPU type is unknown.

EUNKWN produces a formatted dump of the data, showing the relative offset within the data, and the data itself, in word, high-byte, and low-byte formats (all octal), and in binary word format. See Section 4.5 for more information on error logging from unknown devices.

DMPALL

DMPALL is similar to EUNKWN. DMPALL is called if the packet cannot be processed due to an error in format or structure. DISPATCH calls DMPALL if the packet fails any sanity check.

DMPALL produces a formatted dump of the data, showing the relative offset within the data, and the data itself, in word, high-byte, and low-byte formats (all octal), and in binary word format.

Device-Level Modules

Device-level modules contain details of the bit-to-text translation for all supported error-logging devices. DSP2M1/DSP2P1 and DSP3M1/DSP3P1 call them as co-routines. Their names are derived from the names of the associated devices.

See Section 4.5 for information on how these modules are constructed, and how you can write device-level modules for unsupported devices.

Here are the standard error-logging device-level modules:

- EML11 - Processes ML11 errors
- ERK05 - Processes RK05 errors
- ERK67 - Processes RK06 and RK07 errors
- ERL12 - Processes RL01 and RL02 errors
- ERM05 - Processes RM05 errors
- ERM23 - Processes RM02 and RM03 errors
- ERM80 - Processes RM80 errors
- ERP07 - Processes RP07 errors
- ERP23 - Processes RP02 and RP03 errors
- ERP456 - Processes RP04, RP05 and RP06 errors

ERROR LOG CONTROL FILE ARCHITECTURE

- ERS11 - Processes RS11 errors
- ERS34 - Processes RS03 and RS04 errors
- ERX01 - Processes RX01 errors
- ERX02 - Processes RX02 errors
- ET0310 - Processes TS03, TE10, and TU10 errors
- ET1645 - Processes TE16, TU16, and TU45 errors
- ETAll - Processes TAll errors
- ETC11 - Processes TC11 errors
- ETS11 - Processes TS11, TU80 errors
- ETSV05 - Processes TSV05 errors
- ETU58 - Processes TU58 errors
- ETU77 - Processes TU77 errors
- MSCPAT - Processes MSCP Attention errors
- MSCPE - Processes MSCP controller errors
- MSCP60 - Processes MSCP RA60 errors
- MSCP80 - Processes MSCP RA80/RA81 errors
- MSCPEN - Processes MSCP End Packet errors
- MSCPSD - Processes MSCP Small Disk (RC25/RD51/RD50) errors
- MSCPTO - Processes MSCP Timeout errors

Notes Modules

Notes modules contain notes for error conditions that need additional explanation. Notes modules are device-specific and have names derived from the names of the associated device-level module. See Section 4.5.3.2 for more information on how these modules are constructed.

Here are the standard error-logging notes modules:

- NML11 - Processes ML11 notes
- NRK67 - Processes RK06 and RK07 notes
- NRM05 - Processes RM05 notes
- NRM23 - Processes RM02 and RM03 notes
- NT0310 - Processes TS03, TE10, TU10 notes
- NTS11 - Processes TS11, TU80 notes

4.2.2 Program Control Flow

Here is a description of the general flow of program control through the control file modules:

1. RPT opens the control file. In most cases, this is the default control file LX: or LB:[1,6]ERRLOG.ULB. If you wish to use some other filespec, RPT must be rebuilt to prompt for the new name of the file.
2. RPT creates the module table in its dynamic work space. This table contains an entry for each module in the control file universal library.
3. RPT loads the DISPATCH module and transfers control to the ENTRY procedure.
4. The ENTRY procedure is very similar to the root module in most MACRO-11 programs. This procedure declares most of the commonly used data structures. It then enables the ERROR 1 procedure in the ERRORM module as an error handler. Next, it gets and parses the command line by calling the SETUP procedure in the PARSEM module. That done, it performs some general initialization with the INIT 1 procedure from INITM1. ENTRY then sets up a loop which steps through the PACKET_RANGE file, extracting pairs of packet ranges which are fed back to RPT. The next step, performed for each packet range, is to loop through the current packet range, requesting each packet in turn and calling the DISPATCH procedure in the DISPATCH module. When all packets and packet ranges have been requested, it generates summaries (on RSX-11M-PLUS only), if requested, by calling the procedures SUM_ERROR, SUM_GEOMETRY, and SUM_HISTORY from SMRYEP, SMRYGP, and SMRYHP respectively. Finally, to clean up, it calls the FINAL_1 procedure in FINLM1/FINLPL.
5. The DISPATCH procedure in the DISPATCH module declares all of the common subpackets. These are, in order of appearance, HEADER, TASK, DEVICE ID, DEVICE OP, and IO ACTIVITY (conditionally supported on RSX-11M). Each of these subpackets has a mask bit in the HEADER subpacket which indicates the presence of the subpacket. If the bit is set, the subpacket is present and therefore declared. If the bit is not set, the subpacket is not present and consequently not declared. Note that the HEADER subpacket must always be present. As each subpacket is declared, various tests are performed that must be passed or the entire packet is rejected. These tests are for the various selection criteria that the user can specify using command line qualifiers. If the tests are passed, the procedure then computes the name of the appropriate dispatcher module. The dispatcher module name is derived by concatenating the following elements:
 - The string "DSP"
 - The event code (HEADER.CODE_TYPE) converted to ASCII decimal
 - The string "M1" (for RSX-11M) or "P1" (for RSX-11M-PLUS)

For example, an event with a code of 5 would be dispatched to the module DSP5M1 or DSP5P1, depending on the operating system.

ERROR LOG CONTROL FILE ARCHITECTURE

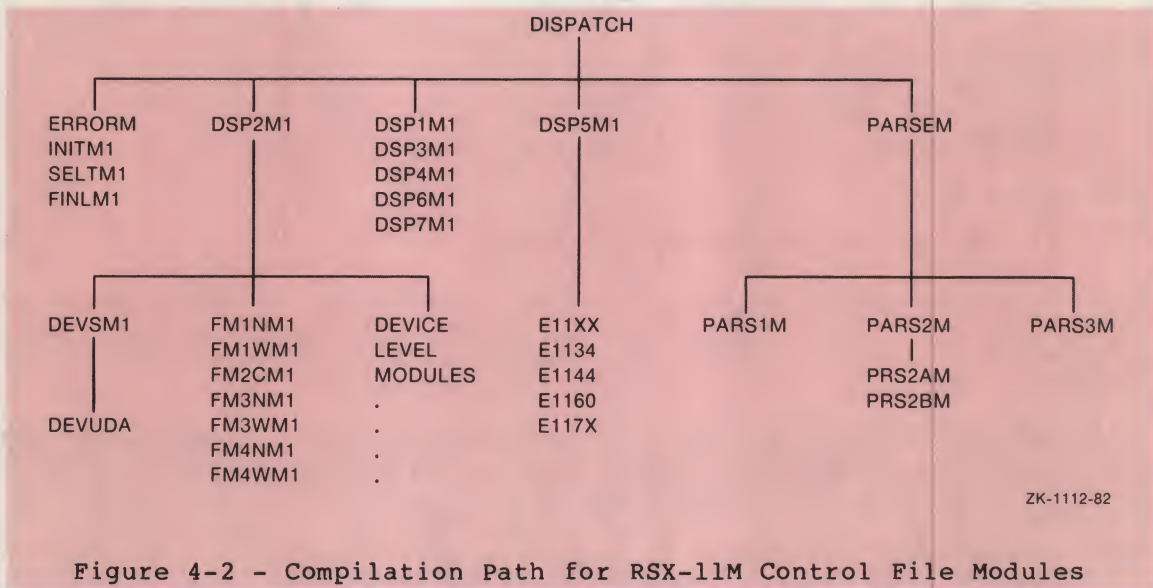
6. The dispatcher modules (or modules they may call) handle the declaration of the DATA subpacket if one is present. The dispatcher modules also perform further selection tests, as appropriate. Eventually, the dispatcher module decides whether or not information about the event should be printed. BRIEF format reports are printed entirely by the dispatcher module. FULL and REGISTER format reports are printed by a combination of:

- One or more of the RSX-11M formatter modules (FM1NM1, FM1WM1, FM2CM1, FM3CM1, FM4NM1, FM4WM1), or one of the RSX-11M-PLUS formatter modules (FMTNP1, or FMTWP1)
- The appropriate dispatcher module
- A device-level module (if it is a device error), or a CPU-level module (if it is a processor or memory error)

4.2.3 Compilation Paths

For both the RSX-11M and RSX-11M-PLUS operating systems, the DISPATCH module must be compiled first. The next modules to be compiled are at the next level, namely (for RSX-11M) ERRORM, DSP2M1, DSP1M1, DSP5M1, and PARSEM using as input the symbol file produced from the compilation of DISPATCH. Modules in the same group, such as ERRORM, INITM1, SELTM1, and FINLM1 all use the same input symbol file (in this case, DISPATCH) and can be compiled in any order.

Figures 4-2 and 4-3 indicate the compilation paths for the RSX-11M and RSX-11M-PLUS modules, respectively.



ERROR LOG CONTROL FILE ARCHITECTURE

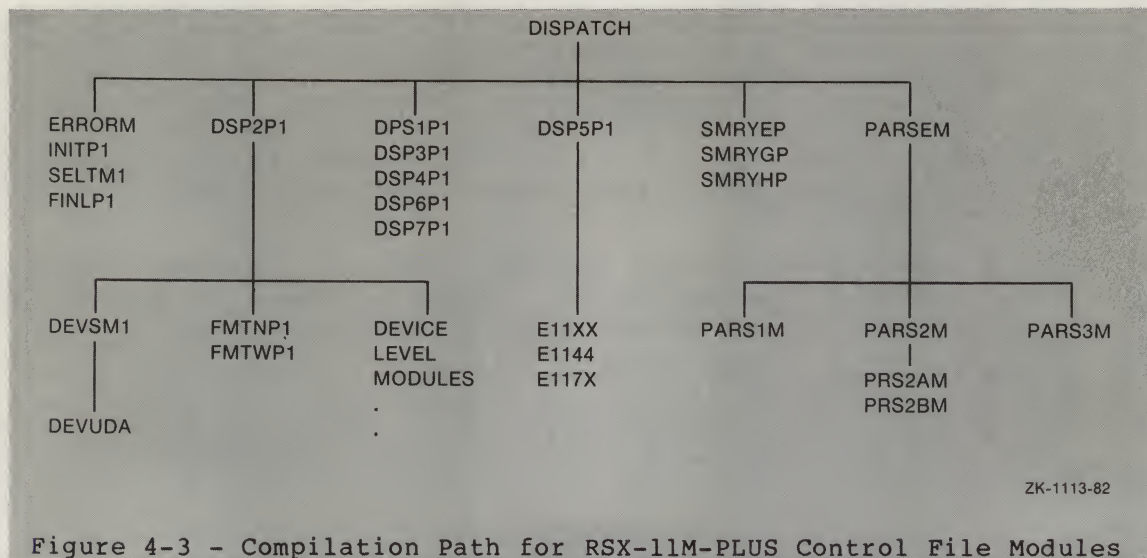


Figure 4-3 - Compilation Path for RSX-11M-PLUS Control File Modules

Where modules in the figure are connected by vertical lines, the upper module is compiled first. The lower module or modules are then compiled using the symbol file produced by the module at the next higher level. Therefore, again using an RSX-11M example, the DSP2M1 module is compiled using the symbol file from DISPATCH, the DEVSM1 module is compiled using the symbol file from DSP2M1, and so on.

Many of the modules in the RSX-11M error log control file have a common source with RSX-11M-PLUS modules. Compile-time conditionals in some of these modules generate variants specific to each operating system. Modules used in the RSX-11M Error Logging System must be compiled using the following compile-time literal declarations:

```

Option> LITERAL SUPPORT.RSX_11M = TRUE
Option> LITERAL SUPPORT.RSX_11M_PLUS = FALSE
Option> LITERAL SUPPORT.IO_ACTIVITY = FALSE
Option> /
    
```

The declaration,

```
LITERAL SUPPORT.IO_ACTIVITY = FALSE
```

can be changed to TRUE to enable processing of I/O activity subpackets. If you choose to do this, you must recompile all control file modules and generate a new system after defining the symbol E\$SACT in RSXMC.

Modules used in the RSX-11M-PLUS Error Logging System must be compiled using the following compile-time literal declarations:

```

Option> LITERAL SUPPORT.RSX_11M = FALSE
Option> LITERAL SUPPORT.RSX_11M_PLUS = TRUE
Option> LITERAL SUPPORT.IO_ACTIVITY = TRUE
Option> /
    
```

See Chapter 5 for a description of the Control File Language used in these declarations.

4.2.4 Modification and Recompilation

You can modify any control file module. After doing so, you must recompile the module and replace it in the control file library.

There is one very important rule to remember when modifying any control file module:

IF

your modification to a module creates new groups, tables, or dynamic tables, OR creates a new variable within any of these structures, OR reorders a variable within any of these structures

THEN

you must also recompile all modules on the same branch of the tree at levels lower than the modified module.

END_IF

Note that recompilation of lower-level modules is not necessary if you modify the run-time logic. For example, modifying the statement

```
IF %STR$LENGTH(PARSE.SWITCH_LIST) EQ 0
```

to

```
IF %STR$LENGTH(PARSE.SWITCH_LIST) EQ 1
```

in the PARSEM module would not require recompiling any of the lower level modules, namely PARS1M, PARS2M, PARS3M, PRS2AM, or PRS2BM.

However, changing the line to read

```
IF %STR$LENGTH(PARSE.SWT_LIST) EQ 1
```

and creating the new variable SWT_LIST in the PARSE group requires recompilation of the lower-level modules. This is because the information in the symbol file consists of group names (in alphabetical order) and the variables defined within the group (in the order declared). The compiler uses the information from the input symbol file to compute relative group and variable numbers for use when a module references a group and/or variable declared in a higher level module. These group and variable numbers, rather than the names, are used to resolve references to groups and variables when a module is loaded. Defining new groups, or variables within a group, changes the relative order of these symbols.

4.3 INTERNAL INTERFACES

This section discusses the specifics of various internal interfaces of the Error Log Control File modules. All of the modules used as examples in this chapter appear at the end of this chapter.

4.3.1 Interaction Between Dispatcher and Device-Level Modules

The following two sections describe, in detail, the interaction between a dispatcher module and a device-level module, using the processing of an RM03 error as an example. Section 4.3.1.1 describes the interaction using an RSX-11M module (DSP2M1). Section 4.3.1.2

describes the differences between the dispatcher modules using an RSX-11M-PLUS module (DSP2Pl).

For this detailed examination, the following discussion refers to the ERM23 device-level module code in Section 4.6.1 and the DSP2M1 dispatcher module code in Section 4.6.2. Both the discussion here and the code in those two sections are keyed to each other by the module names (either ERM23 or DSP2M1) and numbers that look like this: ①.

You may wish to remove the pages for Sections 4.6.1 and 4.6.2 from your book for easier reference in following the interaction between these two modules.

4.3.1.1 Interaction between DSP2M1 and ERM23 - Processing in DSP2M1 begins with DISPATCH having declared all subpackets, except for the DATA subpacket. All subpackets, except for the TASK subpacket, are needed for a device error. DSP2M1 begins by determining that peripheral errors are requested and that the subcode is valid. Having completed these checks, DSP2M1 calls DEVSM1 (DSP2M1 ①). DEVSM1 returns device information in three variables. INTERMOD DEVERR.DISP_NAME contains the name of the device-level module needed to process the DATA subpacket, in this case "ERM23". INTERMOD DEVERR.DRIVE_TYPE contains the string "RM03" for the drive type. The last variable, INTERMOD DEVERR.ALT_NAME contains the string "RM02/03" for the alternate drive type. (The alternate name variable is not used during device error processing.)

After returning from DEVSM1 the NOTE NUMBERS file is cleared (DSP2M1 ②). This deletes any records that may remain there from previous events.

The next step establishes the coroutine relationship with the device-level module (DSP2M1 ③). The DEVICE_ERROR procedure in the DSP2M1 module is one partner while the DEVICE_ENTRY procedure in the ERM23 module is the other. Control passes to the DEVICE_ERROR procedure.

The first thing the DEVICE_ERROR procedure does is pass control to its partner (DSP2M1 ④). Module ERM23 receives control at the beginning of the DEVICE_ENTRY procedure (ERM23 ②).

DEVICE_ENTRY proceeds to declare the DATA subpacket (ERM23 ③). Once this is completed the INTERMOD DEVERR variables are filled in (ERM23 ⑥ and ⑦). All of the variables must be filled in. If the information for a particular variable is unavailable or not applicable, use the string "N/A". For the variable INTERMOD DEVERR.ERROR_CYLINDER the string "???" also has a special meaning; it indicates to DSP2M1 that the section titled Device Error Position Information is to be suppressed.

Once the INTERMOD DEVERR variables are all filled in, DEVICE_ENTRY will coroutine back DEVICE_ERROR (ERM23 ⑧). DEVICE_ERROR regains control where it left off (DSP2M1 ⑤).

DSP2M1 then performs the serial number tests, if required, after having first initialized the variable INTERMOD DEVERR.REJECT_FLAG to FALSE. If the serial number test is failed, the variable INTERMOD DEVERR.REJECT_FLAG is set to TRUE.

The path through the two modules now depends on the report format, either BRIEF, FULL, REGISTER or NONE. Following are explanations of each of these paths.

BRIEF

The REJECT_FLAG variable is tested (DSP2M1 ⑥). If TRUE, nothing is output. If FALSE, one line is output which contains the information required for a BRIEF report. In either case, the variable INTERMOD_DEVERR.PRINT_FLAG is set to FALSE.

FULL

The REJECT_FLAG variable is tested (DSP2M1 ⑦). If TRUE, nothing is output and the PRINT_FLAG variable is set to FALSE.

If REJECT_FLAG is FALSE, the four formatter modules are called to print the information from the common subpackets. The DSP2M1 module then prints (still on the first page) the information passed back in the INTERMOD_DEVERR variables filled in by ERM23. DSP2M1 generates a page break, then prints a header on the second page. When done DSP2M1 sets the PRINT_FLAG variable to TRUE.

REGISTER

The REGISTER path is almost identical to the FULL path. The only difference is that the page containing all of the common information is not printed. The header on the page containing the register translation supplies a summary of the information instead.

NONE

The NONE path sets the PRINT_FLAG variable to FALSE (DSP2M1 ⑧).

All of these paths converge again at DSP2M1 ⑨. At this point control once again passes to the DEVICE_ENTRY procedure in ERM23.

The first thing the DEVICE_ENTRY procedure does upon regaining control is test the PRINT_FLAG variable (ERM23 ⑨). If it is FALSE, the module exits (ERM23 ⑪).

If the PRINT_FLAG variable is TRUE, ERM23 performs the bit-to-text translation of the registers. Following that, any required notes are indicated by PUTs to the NOTE_NUMBERS file specifying the note index (ERM23 ⑩). The module then exits (ERM23 ⑪).

When ERM23 exits DSP2M1 regains control and the coroutine partnership is broken (DSP2M1 ⑩). The DEVICE_ERROR procedure then checks for entries in the NOTE_NUMBERS file. If there are any, DSP2M1 computes the name of the notes file. The name of a notes module is the same as its corresponding device-level module except the first character of the module name is the letter "N", in this case NRM23. The notes module is then called to print the requested notes.

4.3.1.2 Interaction Between DSP2P1 and ERM23 - The relationship and flow of control between the DSP2M1/DSP2P1 module and device-level modules is identical. They both pass the same information and control back and forth at the same points. However, there are differences in the modules themselves. This section explains those differences.

As in the previous discussion, you may wish to remove the pages for Sections 4.6.1 and 4.6.3 from your book for easier reference in following the interaction between these two modules.

ERROR LOG CONTROL FILE ARCHITECTURE

Following are the differences between the DSP2M1 and DSP2P1 modules.

DSP2P1 declares the logical variable INDICATE_TAPE_FLAG (DSP2P1 ①). This variable is set by DEVSM1 to indicate whether or not the device is a magtape. The variable is used later in processing summary information.

If the packet is not rejected and if the report format is not NONE, the variable REPORT.PRINT_COUNT is incremented (DSP2P1 ②). This variable keeps a count of how many events were printed (as opposed to looked at, which is a separate tally).

After the device-level module has printed (if instructed to) and has exited back to DSP2P1, the UPDATE_RECORD procedure in DSP2P1 is called (DSP2P1 ③).

The UPDATE_RECORD procedure tests to see if an ERROR summary was requested (DSP2P1 ④). If not, processing goes on to the GEOMETRY section.

If an ERROR summary was requested, DSP2P1 searches the ERROR_INFO_E file to see if an error having the same error type has been encountered. If so, the record in the file describing that type of error is updated to show that one more error occurred, and when it occurred. If no such error is found in the file, a new record that describes the error is added to the file. Processing then goes on to the GEOMETRY section.

The UPDATE_RECORD procedure then tests to see if a GEOMETRY summary was requested (DSP2P1 ⑤). If not, the procedure exits.

Updating the ERROR_INFO_G file is much the same as updating the ERROR_INFO_E file. The only difference is that the information recorded is somewhat different. In particular, the GEOMETRY summary records information regarding where on the device the error occurred. It is for this reason that we need to know whether or not the device is a magtape; magtapes have no valid geometry information.

4.4 DISPATCHING

This section discusses module dispatching. Dispatching happens at two levels: event-level dispatching and device- or CPU-level dispatching.

4.4.1 Event-Level Dispatching

All events that occur in the Error Logging System are assigned a unique combination of code and subcode. These code/subcode combinations can be found in the file EPKDF.MAC (EPKDF\$ macro in EXEMC.MLB) along with the definition of the structure of error log packets. See Appendix C for a listing of EPKDF\$. Table 4-1 summarizes the error logging code/subcode combinations.

ERROR LOG CONTROL FILE ARCHITECTURE

Table 4-1
Error Logging Code/Subcode Combinations

Code	Subcode
1. Error Log Control	1. Error Log Status Change 2. Switch Logging Files 3. Append File 4. Declare Backup File 5. Show (not logged) 6. Change Limits
2. Device Errors	1. Device Hard Error 2. Device Soft Error 3. Device Interrupt Timeout (hard) 4. Spurious Interrupt (RSX-11M-PLUS only) 5. Device Interrupt Timeout (soft)
3. Device Information	1. Device Information Message
4. Device Control Information	1. Device Mount 2. Device Dismount 3. Device Counts Reset 4. Block Replacement
5. CPU-Detected Errors	1. Memory Error 2. Unexpected Interrupt
6. System Control Information	1. Power Recovery
7. Control Information	1. Time Change 2. System Crash 3. Device Driver Load 4. Device Driver Unload 5. Message

Each code group is processed by one of the dispatcher modules. These modules are named DSP1M1/DSP1P1, DSP2M1/DSP2P1, ..., DSP7M1/DSP7P1. The name of the dispatcher module is derived on the fly in the DISPATCH module's DISPATCH procedure by concatenating the following elements:

- The string "DSP"
- The event code (HEADER.CODE_TYPE) converted to ASCII decimal
- The string "M1" (for RSX-11M) or "P1" (for RSX-11M-PLUS)

The single-digit ASCII conversion of the code value (obtained from the HEADER subpacket) is required because the RSX-11M/M-PLUS Librarian utility LBR allows a maximum of six Radix-50 characters for a module name. The code value 9 is currently unused; values 0 and 8 are reserved.

Once the dispatcher module has been called it checks to see if this type of event was requested. If the event type was not requested, the module returns, effectively ignoring the entry. Event types are requested by using the /TYPE command line qualifier. The event types, codes, and the dispatcher modules that process them, are listed in Table 4-2.

ERROR LOG CONTROL FILE ARCHITECTURE

Table 4-2
Event Types, Codes, and Their Dispatcher Modules

Type	Codes	Dispatcher Modules
ALL	1-7	DSP1M1/DSP1P1, ..., DSP7M1/DSP7P1
CONTROL	1	DSP1M1/DSP1P1
ERRORS	2,3,5	DSP2M1/DSP2P1, DSP3M1/DSP3P1, DSP5M1/DSP5P1
MEMORY	5	DSP5M1/DSP5P1
PERIPHERAL	2,3	DSP2M1/DSP2P1, DSP3M1,DSP3P1
PROCESSOR	5	DSP5M1/DSP5P1
SYSTEM_INFO	4,6,7	DSP4M1/DSP4P1, DSP6M1/DSP6P1, DSP7M1/DSP7P1

Once the dispatcher module determines that this type of event was requested, it checks to see if the subcode is in range. If it is not, the event is rejected with an error message.

At this point dispatcher modules may declare and print the DATA subpacket themselves or may call lower level modules to do so. The error-logging dispatcher modules handle all of the printing for the BRIEF report mode. If the FULL report mode is specified, the dispatcher modules call one or more of the following modules to print the common portions of the event:

System, Width	Formatter Module(s)
RSX-11M, NARROW	FM1NM1, FM2CM1, FM3CM1 and FM4NM1
RSX-11M, WIDE	FM1WM1, FM2CM1, FM3CM1 and FM4WM1
RSX-11M-PLUS, NARROW	FMTNP1
RSX-11M-PLUS, WIDE	FMTWP1

The dispatcher module may print the rest of the event itself or work with a lower-level module.

4.4.2 Device-Level Dispatching

Device-level dispatching is performed with the assistance of the DEVSM1 module. This module is called by DSP2M1/DSP2P1, DSP3M1/DSP3P1 and DSP4M1/DSP4P1 and determines, among other things, the correct device-level module for the event.

Here is a description of how DEVSM1 works (see the source code for exception cases; this discussion addresses only usual cases). The first thing DEVSM1 checks is whether there is a DEVICE_ID subpacket. If no DEVICE_ID subpacket is found, an error results. Once past that check, DEVSM1 uses the device mnemonic to search the DEVICE_INFO table. If the device is not found, DEVSM1 specifies the EUNKWN module in the variable INTERMOD_DEVERR.DISP_NAME.

Assuming that the mnemonic is recognized, DEVSM1 tests to see if (a) the mnemonic is that of a MASSBUS device, and (b) there is a DATA subpacket. Assuming both are true, DEVSM1 looks ahead into the DATA subpacket to obtain the MASSBUS Drive Type from the logged registers.

This value is unique for each MASSBUS device. Once this value is obtained, the DEVICE_INFO table is searched again, this time using the drive-type value as the key. Assuming this search turns up a match, the variable INTERMOD_DEVERR.DISP_NAME is filled in with the module name specified by the resulting record in the table.

If there is no DATA subpacket, or if the device is not a MASSBUS device, the search of the table ends up pointing to the first record that matched on the specified mnemonic. DEVSM1 performs a further search of the table based on the mnemonic as well as the device size (which is provided in the variable DEVICE_ID.DEV_TYPE). The variable INTERMOD_DEVERR.DISP_NAME is then filled in with the module name specified in the record that is the result of this search.

4.4.3 CPU-level Dispatching

CPU-level dispatching is performed by DSP5M1/DSP5P1. The HEADER subpacket contains a variable called PROC_TYPE that indicates the type of processor the error was logged on. DSP5M1/DSP5P1 uses that variable to search a table that contains module names associated with the CPU-type value.

4.5 SUPPORT OF NON-DIGITAL DEVICES

This section explains what you have to do to provide error-logging support for non-DIGITAL devices.

Adding error-logging support for a non-DIGITAL device consists of either one or three steps, depending on the desired level of support. The first step is to include error-logging support in the driver. Without this support no information can be logged for the device. For full error-logging support, you must perform two more steps: write a device-level module for the new device, and add it to the control file library and make the Error Logging System aware of the new module.

The following sections show you what you need to accomplish these steps.

4.5.1 Error-logging of Unknown Devices

The Error Logging System can handle entries from devices unknown to the system. Entries from an unknown device are handled by the EUNKWN module, which functions as a universal device-level module. For a BRIEF report, EUNKWN will pass back "N/A" in the INTERMOD_DEVERR variables to indicate that the information is not available. For a FULL report, EUNKWN prints the device registers in a dump-style format where the bit-to-text translation would normally take place. The rest of the report is unchanged.

4.5.2 Providing Driver Support for a Non-DIGITAL Device

The Executive module ERROR contains the routines to be used by a driver to log device errors. A device error in this sense can be a real error, a timeout, or perhaps an informational message. The following sections discuss the routines in general. See the code in [11,10]ERROR.MAC for more detail.

ERROR LOG CONTROL FILE ARCHITECTURE

For the most part, driver support is the same for RSX-11M and RSX-11M-PLUS. Where there are differences, the full discussion is repeated.

4.5.2.1 \$BMSET on RSX-11M - On RSX-11M, the \$BMSET coroutine raises the processor priority to 7 (to lock out interrupts) and then calls the caller to start the I/O function. When the re-called caller returns, \$BMSET lowers the processor priority to 0, thus allowing interrupts once again.

INPUTS

None

OUTPUTS

None

4.5.2.2 \$BMSET on RSX-11M-PLUS - On RSX-11M-PLUS, the \$BMSET coroutine raises the processor priority to 7 (to lock out interrupts), sets the "interrupt active" bit S2.ACT in S.ST2 of the SCB, and then calls the caller to start the I/O function. When the re-called caller returns, \$BMSET lowers the processor priority to 0, thus allowing interrupts once again.

INPUTS

R4 = SCB Address

OUTPUTS

The "interrupt active" bit S2.ACT is set

4.5.2.3 \$DVTMO and \$DTOER on RSX-11M - On RSX-11M, the routine \$DVTMO logs device timeouts at PR0, and the routine \$DTOER logs device timeouts at device priority. The routines behave identically, except that \$DTOER disables the device interrupt and lowers the processor priority to 0.

If the symbol D\$\$IAG is defined, the routines test to see if the timeout is a diagnostic function. Diagnostic functions are never logged.

The routines load the error code and subcode in R0 and the routine falls into the routine \$DVCER.

INPUTS

R2 = CSR Address

R4 = SCB Address

OUTPUTS

R1 = I/O Packet Address (if D\$\$IAG defined and a diagnostic function)

C = 0, if D\$\$IAG not defined or not a diagnostic function. Create an error log packet and fill it in. Put a pointer to the packet (S.BMSV) in the SCB and set the "error in progress" bit SP.EIP in S.PRI.

ERROR LOG CONTROL FILE ARCHITECTURE

C = 1, If D\$\$IAG defined and a diagnostic function. Set the "error in progress" bit SP.EIP in S.PRI. Do not create an error log packet.

4.5.2.4 \$DVTMO and \$DTOER on RSX-11M-PLUS - On RSX-11M-PLUS, the routine \$DVTMO logs device timeouts at PR0, and the routine \$DTOER logs device timeouts at device priority. The routines behave identically except that \$DTOER disables the device interrupt and lowers the processor priority to 0.

The routines clear the "interrupt active" bit S2.ACT in the SCB word S.ST2. They then test to see if the timeout is a diagnostic function. Diagnostic functions are never logged.

The routines load the error code and subcode in R0 and the routine falls into the routine \$DVCER.

INPUTS

R2 = Address of a block of registers to log (must be the CSR address if KS.MBC is set)
R4 = SCB Address
R5 = UCB Address

OUTPUTS

R0 = IE.DNR and 377 (Device not Ready)
R1 = I/O Packet Address

C = 0, if not a diagnostic function. Create an error log packet and fill it in. Put a pointer to the packet (S.BMSV) in the SCB and set the "error in progress" bit SP.EIP in S.PRI.

C = 1, If a diagnostic function. Do not create an error log packet.

4.5.2.5 \$DVERR and \$DVCER on RSX-11M - \$DVERR and \$DVCER are the same; \$DVCER is the routine name, and \$DVERR is a synonym. This routine logs device errors. If an error is already in progress on the device, it will be ignored. If not, \$DVCER allocates an error log packet and fills it in with the context of the current transfer. Note that this routine requires that there be an I/O packet associated with this error. See the routine \$LOGGER (Section 4.5.2.9) to log an error where there is no I/O active on the device.

INPUTS

R4 = SCB Address
R5 = UCB Address

OUTPUTS

If no error is already in progress on this device, allocate an error log packet, fill it in, point the SCB to the packet, and set the "error in progress" bit.

If an error is in progress on this device, this routine is a no-op.

ERROR LOG CONTROL FILE ARCHITECTURE

4.5.2.6 \$DVERR and \$DVCER on RSX-11M-PLUS - \$DVERR and \$DVCER are the same; \$DVCER is the routine name, and \$DVERR is a synonym. This routine logs device errors. If an error is already in progress on the device, it will be ignored. If not, \$DVCER allocates an error log packet and fills it in with the context of the current transfer. Note that this routine requires that there be an I/O packet associated with this error. See the routine \$LOGGER (Section 4.5.2.9) to log an error where there is no I/O active on the device.

Note that, on RSX-11M-PLUS, information about concurrent I/O activity on other devices is also logged.

INPUTS

R2 = Address of a block of registers to log (must be the CSR address if KS.MBC is set)
R4 = SCB Address
R5 = UCB Address

OUTPUTS

If no error is already in progress on this device, allocate an error log packet, fill it in, point the SCB to the packet, and set the "error in progress" bit.

If an error is in progress on this device this routine is a no-op.

4.5.2.7 \$NSIER - This routine logs nonsense interrupts. The routine identifies the interrupting vector and logs the error.

INPUTS

@(SP) = Contains bits 06:04 of the unused vector number.

OUTPUTS

If a nonsense interrupt is in the process of being logged, increment the interrupt count.

If this is the beginning of the processing of a nonsense interrupt, identify the vector and create and queue an error log packet.

4.5.2.8 \$FNERL - This routine is called at I/O completion, or when it is necessary to queue an error log packet after a successful recovery of a mid-transfer error. In essence, this routine completes the processing of an error.

The routine first inserts the error retry information. It then tests to see if this was a hard (unrecoverable) error or a soft (recoverable) error, and updates the packet accordingly. All errors are assumed to be "hard" up to this point. Depending on the result of that test, \$FNERL tests against the appropriate limit to see if the limit has already been met. If the limit had been previously met the packet is discarded. If not, \$FNERL updates the appropriate error count, logs the packet, and sets the SCB to show that the processing of this error has been completed.

ERROR LOG CONTROL FILE ARCHITECTURE

INPUTS

R0 = First I/O Status word
R2 = Starting and Final error retry counts (if 0, do not update limits)
R3 = Error Log Packet Address (if R4 = 0)
R4 = SCB Address or 0
R5 = UCB Address

OUTPUTS

Either queue or discard the error log packet (depending on the limits) and set the SCB to indicate that no error is being processed.

4.5.2.9 \$LOGGER - Drivers use \$LOGGER to create an error log packet when no I/O is present, such as when a driver receives an unsolicited interrupt from a device that contains information that should be logged. \$LOGGER creates the packet normally, but the driver is responsible for filling in the DATA subpacket information. Otherwise, processing is similar to \$DVERR.

INPUTS

R1 = Length of data to be logged (in bytes)
R4 = SCB Address (If 0, then no I/O packet is present)
R5 = UCB Address

OUTPUTS

C = 1, Error cannot be logged for some reason
C = 0, Error can be logged
R1 = Address of DATA area in the packet
R3 = Address of Error Log packet

4.5.2.10 LOGTST - This routine is not for use by drivers. Other routines in the ERROR module call LOGTST to see if an error can or should be logged.

4.5.2.11 \$CRPKT - This routine creates an error log packet. It is called as part of the \$SMMSG directive processing, and by other Executive routines as part of the processing of a memory error, nonsense interrupt, time change, power fail recovery, or device error.

In general, the routine determines the required format and size of the packet, allocates the required amount of pool, and then fills in the packet. It obtains information from SYSCOM, the appropriate DCBs, UCBs, SCBs, TCBs, VCBs, and the I/O packet, as required.

Note that a HEADER subpacket is always required. A forced system crash will result if \$CRPKT detects the condition of "no HEADER subpacket".

Note also that on RSX-11M, information about concurrent I/O activity on other devices can also be optionally logged. Do this by defining the symbol E\$SACT and doing a new system. (You must recompile the error-logging control files, as well.)

ERROR LOG CONTROL FILE ARCHITECTURE

On RSX-11M-PLUS, information about concurrent I/O activity on other devices is always logged in addition to the activity on the device in question.

INPUTS

R0 = Packet Code and Subcode (See EPKDF for details)
R1 = Length of DATA subpacket
R2 = Control Mask word (See EPKDF for details)
R3 = Beginning Address of data for DATA subpacket
R4 = TCB Address (for TASK subpacket)
R5 = UCB Address (for DEVICE IDENTIFICATION subpacket)

OUTPUTS

R0 = Unchanged
R1 = Beginning Address of data in the DATA subpacket
R2 = Unchanged
R3 = Beginning Address of Error Log packet
R4 = Unchanged

C = 0, A packet was created
C = 1, A packet was not created
R5 = Unchanged

4.5.2.12 CALDEV on RSX-11M-PLUS - On RSX-11M-PLUS, CALDEV calculates the logical unit number for the given UCB.

INPUTS

R0 = Pointer into the Error Log packet
R3 = DCB Address
R5 = UCB Address

OUTPUTS

Unit number stored in the Error Log packet at the byte (R0)

R0 = Updated to point to next byte in Error Log packet

4.5.2.13 \$QUPKT - This routine queues an Error Log packet. If there is no other packet in the queue, \$QUPKT requests the error logger task (ERRLOG) with a delay of 2 seconds. If there is another entry in the queue, \$QUPKT requests ERRLOG to run immediately. Command packets (from ELI) always cause ERRLOG to run immediately.

INPUTS

R3 = Pointer to packet for insertion in queue

OUTPUTS

None

4.5.2.14 **\$QERMV** - This routine removes an entry from the error log queue and transfers it to a user buffer. It is called only by ERRLOG.

INPUTS

R4 = Length of user buffer
R5 = Address of user buffer

OUTPUTS

R1 = Length of packet
R4 = Unchanged
R5 = Unchanged

C = 0, Packet was successfully removed

C = 1, Either no packet to remove or packet was too long. If R1 <> 0, the packet was too long and R1 contains the packet length. If R1 = 0, then there was no packet to remove.

4.5.3 Error-Logging Support for a Non-DIGITAL Device

Full error-logging support requires two steps beyond driver support. The first step is to write the device-level module for the new device. This module contains the detailed instructions on how to interpret the logged information, that is, the bit-to-text translation information for the device registers. The information common to all events is interpreted by the DIGITAL-supplied modules.

The second step is to add the new module to the control file library and make the Error Logging System aware of the new module. The following sections explain these steps in detail.

4.5.3.1 **How to Write a Device-Level Module** - This section explains the general structure of device-level modules, using the RM02/03 module ERM23 as an example. Section 4.6.1 is an annotated listing of ERM23; Section 4.6.4 is an annotated listing of the notes module for the RM02/03. Both the discussion here and the code in those two sections are keyed to each other by the module names (either ERM23 or DSP2M1) and numbers that look like this: ① .

You may wish to remove the pages for Sections 4.6.1 and 4.6.4 from your book for easier reference in following the interaction between these two modules.

In general, the flow of a device-level module proceeds as follows:

- ① MODULE statement followed by module header
- ② PROCEDURE statement
- ③ SUBPACKET declaration
- ④ Register definitions
- ⑤ Declaration of local work variables and table declarations
- ⑥ Intermodule variable loading
- ⑦ Error-type determination

- ⑧ Coroutine back to caller
- ⑨ Bit-to-text translation and register printing
- ⑩ Note requirements indicated
- ⑪ Exit the module.

Each of these procedures are described in the following sections.

4.5.3.1.1 MODULE Statement -

①

The MODULE statement for a device-level module must be of the form:

```
MODULE modulename 'ident' ;
```

The module_name must match the name specified for this device in the DISP_NAME field of the DEVICE_INFO table in the DEVSM1 module (See Section 4.5.3.4). Generally, the module name begins with the letter "E", followed by five or fewer letters indicating the device or devices served by the module. For example, the ERM23 module handles the RM02 and RM03 disks, while the ERP456 module serves the RP04, RP05 and RP06 disks.

The ident field is exactly what it implies, an identification value that is stored in the module. Generally, the ident begins with a letter that identifies the operating system the module is intended to be used with, such as "M", followed by a version and update number in the standard DIGITAL style.

The module header follows. This includes the copyright statement, author, date written, and audit trails of modifications.

4.5.3.1.2 PROCEDURE Statement -

②

The PROCEDURE statement for a device-level module must be of the form:

```
PROCEDURE DEVICE_ENTRY
```

The procedure name must be DEVICE_ENTRY. This name is hard coded into the DSP2M1/DSP2P1 and DSP3M1/DSP3P1 dispatcher modules.

4.5.3.1.3 SUBPACKET Declaration -

③

The device-level module is responsible for the declaration of the device data (usually in the form of registers). The SUBPACKET declaration defines the number of registers, how they are printed, and the bit-to-text translations for the various bits and fields of the registers. The general format of the statements is as follows:

ERROR LOG CONTROL FILE ARCHITECTURE

```
SUBPACKET subpacket_name = DISP.NEXT_PACKET NAMED ;
```

```

    reg_name:    WORD MACHINE ;
    :           BIT [15]:      'true_text' ;
    :           BIT [14]:      'true_text' ;
    :           :              'false_text' ;
    aux_label:   FIELD [12:2]:  'Bits 12 and 13 = '
    :           :              | %CNV $BINARY(Subpacket_name.aux_label, 2, '0')
    :           :              | ' (B)' ;
    :           BIT [11]:      'true_text' ;
    :           .
    :           .
    :           .
    reg_name:    WORD MACHINE ;
    :           .
    :           .
    :           .
END_PACKET ;

```

The subpacket name is usually REGISTER, although this name is not required. DISP.NEXT_PACKET is a variable that contains the subpacket number of the 'data' subpacket and has been set up by the preceding modules. The NAMED qualifier indicates to RPT that the register labels are to be saved for later printing.

What follows next are the definitions of the registers and their bits and fields.

The end of the subpacket declaration is indicated by the statement 'END_PACKET ;'.

4.5.3.1.4 Register Definitions -

4

The label assigned to a register provides both a reference to the register (a variable name) and a name for the register when printing. The register name is printed later on (if you specified a FULL format report). In most cases, the Error Logging System uses the same register names used by DIGITAL field service hardware documents.

For the RM02/03, the first register declared looks like:

```

RMCS1:    WORD MACHINE ;
:         BIT [15]:      '*Special Condition set' ;
RMCS1_TRE: BIT [14]:      '*Transfer Error' ;
:         BIT [13]:      '*MASSBUS Control Bus Parity Err' ;
:         BIT [12]:      '*Unused bit set' ;
:         BIT [11]:      ' Drive Available',
:         :              '*Drive not Available (other port using it)' ;
:         BIT [10]:      ' UNIBUS B Selected for Data Transfer',
:         :              ' UNIBUS A Selected for Data Transfer' ;
RMCS1_BA: FIELD [8:2]:    ' BA17,BA16 = '
:         :              | %CNV $BINARY(REGISTER.RMCS1_BA, 2, '0')
:         :              | ' (B)' ;
:         BIT [ 7]:      ' Controller Ready',
:         :              ' Controller not Ready' ;
:         BIT [ 6]:      ' Interrupt Enabled',
:         :              ' Interrupt not Enabled' ;
:         .
:         .
:         .

```


ERROR LOG CONTROL FILE ARCHITECTURE

The first line indicates that the name of the register is RMCS1 and that it is a WORD in length (16 bits). The MACHINE qualifier states that, when printed, the register is to be formatted in the native radix for the machine that the report is being generated on. The native radix for the PDP-11 is octal, and for the VAX-11, hexadecimal. Other print qualifiers are available to change the radix, such as HEX, OCTAL, DECIMAL, BCD, BINARY, and RAD50.

The second line defines bit 15 of the register RMCS1, including when it is to be printed and what is to be printed. Only one text string is provided. This indicates that the bit is to be printed only when true (set). Otherwise, nothing is printed for that bit.

Bit 14 has a label of RMCS1_TRE. Labels assigned to bits and fields are never printed. They are allowed so you can reference the bit or field as a variable. As with bit 15, the text for this bit is printed only if the bit is set.

Bit 11 has two text arguments. The first argument is printed if the bit is set and the second argument is printed if the bit is reset. In other words, this bit will always be printed.

Bits 8 and 9 are defined to be a FIELD with the variable name RMCS1_BA. The arguments for a field are as follows:

```
FIELD [starting_bit_number:number_of_bits]: 'other_string',
                                             '0_string',
                                             '1_string',
                                             '2_string',
                                             .
                                             .
                                             'N_string' ;
```

The 0_string is printed if the value of the field is zero. The 1_string is printed if the value of the field is 1, and so on. The other_string is printed if the field has a value that has no corresponding text string. Note that for the field RMCS1_BA there is only an other_string. Therefore, this field is always printed.

A technique that is used in the DIGITAL device-level modules is to declare a field over any contiguous unused bits. The other_string is defined to be 'Unused bits set', and the 0_string is defined to be NULL (the null, or zero length string). If the field has the value zero, nothing is printed. If, however, any of the bits are set, the field appears in the report.

Note that all of the text strings associated with bits and fields have as their first character either a space or an asterisk. RPT, when printing the text for a bit or field, removes the first character of the string and places it in front of the bit or field position indicator. An asterisk signals some kind of special condition. For example, bit 11 of RMCS1 can print one of two ways, either as:

```
[11] Drive Available
```

or as:

```
*[11] Drive not Available (other port using it)
```

Remember that the asterisk does not necessarily indicate an error, just something interesting. A blank in front of the position indicator means a normal or status condition.

ERROR LOG CONTROL FILE ARCHITECTURE

You can use IF...THEN...ELSE, CASE, and SELECT statements to conditionalize the declaration of the subpacket. The statement blocks in these structures must be enclosed by BEGIN and END. You can use variables previously declared in the subpacket even though the declaration of the subpacket is not complete. Also note the use of the %LOK (lookahead) functions in various device-level modules. They look into a subpacket before it is declared, usually to produce variables to control the declaration.

Note the variable REGISTER.LENGTH towards the end of the subpacket declaration in ERM23. This variable was created when the SUBPACKET statement was executed. The variable name is of the form subpacket_name.LENGTH and contains the number of bytes in the subpacket.

4.5.3.1.5 Declaration of Local Work Variables and Tables -

5

The device-level module often needs some local variables and tables. These are generally defined after the end of the subpacket declaration, although this is not required. Remember, however, that variables must be declared in a module before they can be used.

4.5.3.1.6 Loading of the Intermodule Variables -

6

The DISPATCH module declares a collection of variables having the group name INTERMOD_DEVERR. Some of these ASCII string variables pass information from the device-level modules back to their caller. The variables that must be filled in are:

- INTERMOD_DEVERR.DRIVE_SN
- INTERMOD_DEVERR.DEV_FUNCTION
- INTERMOD_DEVERR.PHYS_UNIT
- INTERMOD_DEVERR.ERROR_CYLINDER
- INTERMOD_DEVERR.ERROR_SECTOR
- INTERMOD_DEVERR.ERROR_HEAD
- INTERMOD_DEVERR.ERROR_GROUP
- INTERMOD_DEVERR.BLOCK_NUMBER
- INTERMOD_DEVERR.ERROR_TYPE
- INTERMOD_DEVERR.DRIVE_TYPE (See Section 4.5.3.3 for more details on this variable.)

This section of the module is where these variables are filled in. Use the string 'N/A' if the information is either not applicable or not available. Note that for certain devices, most notably magnetic tapes, the ERROR CYLINDER variable is filled in with the string '???'. This flag tells the dispatcher module to suppress the printing of the section entitled Device Error Position Information. Note that one of the variables to be filled in contains the error type. See the next section for more details on how the error type is determined.

4.5.3.1.7 Determination of the Error Type -

7

The error-type definition is essentially a determination of the most likely problem as indicated by the error bits for a given event. It is not a determination of 'what broke', but rather an indication of 'what happened'. The error type is determined solely on the basis of the bits in the current event. No inter-event analysis is performed.

The error type is determined by a precedence parse of the various error bits found in the device registers. The DECODE statement, in conjunction with IF...THEN...ELSE-type constructs, is used to search the bits in a specific order. The first condition found to be true stops the search.

4.5.3.1.8 Coroutine Back to Caller -

8

Once all of the intermodule variables have been filled in, a coroutine statement returns control to the device module's caller. The caller examines the returned information and determines whether to continue processing the event. Nothing has been printed up to this point in the processing of this event.

If the decision is not to proceed (to reject the event), the caller (a) sets the variable INTERMOD_DEVERR.PRINT_FLAG to FALSE and, (b) coroutines back to the device-level module.

If the decision is to proceed, the caller performs some or all of the printing, depending on whether the print format is FULL or BRIEF. If the FULL format is specified, the caller (a) prints everything except the device registers, (b) sets the variable INTERMOD_DEVERR.PRINT_FLAG to TRUE, and (c) coroutines back to the device-level module. If the format is BRIEF, the caller (a) performs all required printing, (b) sets the variable INTERMOD_DEVERR.PRINT_FLAG to FALSE and (c) coroutines back to the device-level module.

When the device-level module regains control it examines the print flag. If TRUE, the module prints the device registers and generates any required note indicators. If the print flag is FALSE, the module exits.

4.5.3.1.9 Perform the Bit-To-Text Translation and Register Printing -

9

If the variable INTERMOD_DEVERR.PRINT_FLAG is TRUE the device-level module prints the device registers and performs the required bit-to-text translation. This is done by executing a WRITE statement (to produce column headers) followed by a WRITE_GROUP statement. The WRITE_GROUP statement references the subpacket name specified in the SUBPACKET statement. It also uses two variables, REPORT.W_G_F_1 and REPORT.W_G_F_2, as format strings. These variables are initialized by the INITM1 module and contain the format strings for printing the register data in either WIDE or NARROW format. If you need to print data that does not conform to the formats defined by these variables, you can define your own format. You can test the logical variable REPORT.WIDE to determine whether a WIDE or NARROW report was requested.

ERROR LOG CONTROL FILE ARCHITECTURE

If the variable `INTERMOD_DEVERR.PRINT_FLAG` is `FALSE`, the device-level module exits.

4.5.3.1.10 Indicate Any Notes that are Required -

⑩

The Error Logging System can print notes for certain conditions that need additional explanation. If you need such notes, you can create a notes module (See Section 4.5.3.2 for details) and include it in the library. You can then request a note by referencing it from the device-level module.

You request a note by performing a `PUT` into the `NOTE_NUMBERS` file specifying the note number in the `NOTE_NUMBERS.INDEX` variable. For example, the RM02/03 device-level module can optionally generate a note if certain unused bits in the RMDA register are set. This is done with the code:

```
!
! If the unused bits 5 to 7 are set in the RMDA register.
!
IF (REGISTER.RMDA [5:3] NE #BB'0')
THEN
    !
    ! Print the note saying that it may cause an invalid
    ! sector address to be recognized resulting in a
    ! possible invalid address error.
    !
    PUT NOTE_NUMBERS INDEX = 1 ;
END_IF ;
```

When the device-level module exits, the caller tests to see whether any notes were requested. If notes were requested, the dispatcher strips the first character from the device-level module's name and replaces it with the letter 'N'. For example, the notes module for ERM23 (the RM02/03 device-level module) is NRM23. The dispatcher calls the notes module, which determines which notes were requested and prints them.

Multiple notes can be requested. They are printed in the order requested.

4.5.3.1.11 Exit the module -

⑪

When everything is done, the device-level module exits. Exiting a module implies a `RETURN` to the module's caller. Exiting from a device-level module also breaks the coroutine relationship.

4.5.3.2 How to Write a Notes Module - This section explains the structure of a notes module using the RM02/03 notes module as an example. Section 4.6.4 contains an annotated listing of this module.

ERROR LOG CONTROL FILE ARCHITECTURE

Here, in general, is the flow of a notes module:

- ① MODULE statement followed by module header
- ② PROCEDURE statement
- ③ Notes heading
- ④ Selection of a note for printing
- ⑤ Handling of an unknown note number
- ⑥ Getting the next note
- ⑦ Exit the module

These sections are now explained in detail.

4.5.3.2.1 MODULE Statement -

①

The MODULE statement for a notes module must be of the form:

```
MODULE module_name 'ident' ;
```

The module_name of a notes module is related to its corresponding device-level module name by replacing the first letter of the device-level module's name with the letter 'N' to get the notes module name. This convention must be followed, because the notes module name is derived from the name of the device-level module and is never looked up in a table.

See Section 4.5.3.1 for an explanation of the 'ident' field of the MODULE statement.

4.5.3.2.2 PROCEDURE Statement -

②

The PROCEDURE statement for a notes module must be of the following form:

```
PROCEDURE NOTES
```

The procedure name must be NOTES. This is wired into the DSP2M1/DSP2P1 and DSP3M1/DSP3P1 dispatcher modules.

4.5.3.2.3 Notes Heading -

③

The notes heading declares what is about to be printed. Notice that notes appear directly following the register interpretation in FULL and REGISTER reports only.

4.5.3.2.4 Selecting a Note for Printing -

4

Notes are selected for printing by testing the NOTE_NUMBERS file for context after performing a POINTER NOTE_NUMBERS FIRST operation. If records remain (that is, if there is context) a SELECT is performed on the variable NOTE_NUMBERS.INDEX. This variable indicates which note to print.

4.5.3.2.5 Handling an Unknown Note Number -

5

The ELSE clause of the SELECT statement traps unknown note numbers. A SIGNAL is performed using the 'UNKNWNNOT' error indication. The note number and the drive type are passed to the error handler as string arguments.

4.5.3.2.6 Getting the Next Note -

6

The next note is obtained by POINTER NOTE_NUMBERS NEXT. This causes RPT to point to the next record in the NOTE_NUMBERS file. If another record exists, the NOTE_NUMBERS file has context at the top of the WHILE...DO loop; otherwise there will be no context, which means that there will be no more notes.

4.5.3.2.7 Exit the Module -

7

When everything is done, the notes module exits. Exiting a module implies a RETURN to the module's caller.

4.5.3.3 MASSBUS and Non-MASSBUS Considerations - All device-level modules work essentially the same way. The only exception is that MASSBUS modules are not required to fill in the variable INTERMOD_DEVERR.DRIVE_TYPE, whereas non-MASSBUS modules are.

This exception has to do with mixed MASSBUS configurations. With mixed configurations, the Executive's database may not match the actual configurations. A mismatch can happen if unit plugs have been inadvertently swapped.

The Error Logging System deals with this possibility as follows:

1. When a device's mnemonic is found in the DEVICE_INFO table in module DEVSM1, the MASSBUS_FLAG is checked. If it is TRUE, a lookahead into the device registers returns the device's DRIVE_TYPE.
2. The DEVICE_INFO table is then searched again to find a record having that drive type.

ERROR LOG CONTROL FILE ARCHITECTURE

3. The Error Logging System then dispatches to the module corresponding to the actual registers logged, not to the module indicated by the mnemonic provided by the Executive.

For MASSBUS devices, the Error Logging System uses the device name provided by the DEVICE_INFO table. This name will always be correct, as each MASSBUS device has a unique drive-type value. If there is a mismatch between the mnemonic supplied and the device type as determined by examining the registers, the device-type field in the printed report is preceded by an asterisk.

For non-MASSBUS devices, it is the device-level module's responsibility to supply correct drive-type information. The DEVSM1 module fills in the value based on the device's mnemonic and size, but sometimes this information is not accurate. The RK03 and RK05 are examples of where this is necessary. Both RK03 and RK05 device errors are processed by the ERK05 module. The ERK05 module figures out, based on the device registers, which kind of drive it is and fills in the DRIVE_TYPE variable accordingly. Another example is DU devices. In this case, the Error Logging System is only concerned that the device mnemonic is DU. It is up to the modules that handle these devices to provide the drive-type information.

4.5.3.4 Making the New Device-Level Module Known - The Error Logging System is made aware of a new device-level module by adding a record to the DEVICE_INFO table in the DEVSM1 module. A section of the table is reproduced in Table 4-3.

Table 4-3
The DEVICE_INFO Table

TABLE DEVICE_INFO ;					
MNEMONIC	:ASCII [2] ;				! Device mnemonic
PRINT_NAME	:ASCII [6] ;				! Name for printing
ALT_PRINT_NAME	:ASCII [12] ;				! Alternate name for printing
DISP_NAME	:ASCII [6] ;				! Name of device module
SIZE	:LONGWORD ;				! Size of device
MASSBUS_FLAG	:LOGICAL ;				! True if a MASSBUS device
DRIVE_TYPE	:BYTE ;				! MASSBUS device type number
BEGIN_TABLE					
'CT', 'TU60',	'TU60',	'ETA11',	#LD'0',	FALSE,	#BO'0' ;
'DB', 'RP04',	'RP04/05',	'ERP456',	#LD'171798',	TRUE,	#BO'20' ;
'DB', 'RP05',	'RP04/05',	'ERP456',	#LD'171798',	TRUE,	#BO'21' ;
'DB', 'RP06',	'RP06',	'ERP456',	#LD'340670',	TRUE,	#BO'22' ;
'DD', 'TU58',	'TU58',	'ETU58',	#LD'512',	FALSE,	#BO'0' ;
'DF', 'RF11',	'RF11',	'ERS11',	#LD'-1',	FALSE,	#BO'0' ;
'DK', 'RK05',	'RK03/05',	'ERK05',	#LD'4800',	FALSE,	#BO'0' ;
'DL', 'RL01',	'RL01',	'ERL12',	#LD'10240',	FALSE,	#BO'0' ;
'DL', 'RL02',	'RL02',	'ERL12',	#LD'20480',	FALSE,	#BO'0' ;
'DM', 'RK06',	'RK06',	'ERK67',	#LD'27126',	FALSE,	#BO'0' ;
'DM', 'RK07',	'RK07',	'ERK67',	#LD'53790',	FALSE,	#BO'0' ;

(continued on next page)

ERROR LOG CONTROL FILE ARCHITECTURE

Table 4-3 (Cont.)
The DEVICE_INFO Table

'DP', 'RP03',	'RP03',	'ERP23',	#LD'80000',	FALSE, #BO'0' ;
'DR', 'RM02',	'RM02/03',	'ERM23',	#LD'131680',	TRUE, #BO'25' ;
'DR', 'RM03',	'RM02/03',	'ERM23',	#LD'131680',	TRUE, #BO'24' ;
'DR', 'RM05',	'RM05',	'ERM05',	#LD'500384',	TRUE, #BO'27' ;
'DR', 'RM80',	'RM80',	'ERM80',	#LD'242606',	TRUE, #BO'26' ;
'DR', 'RP07',	'RP07',	'ERP07',	#LD'1008000',	TRUE, #BO'42' ;
'DS', 'RS03',	'RS03/04',	'ERS34',	#LD'1024',	TRUE, #BO'0' ;
'DS', 'RS03',	'RS03/04',	'ERS34',	#LD'1024',	TRUE, #BO'1' ;
'DS', 'RS04',	'RS03/04',	'ERS34',	#LD'2048',	TRUE, #BO'2' ;
'DS', 'RS04',	'RS03/04',	'ERS34',	#LD'2048',	TRUE, #BO'3' ;
'DT', 'TU56',	'TU56',	'ETC11',	#LD'576',	FALSE, #BO'0' ;
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

The columns of the table, taken from left to right, correspond to the declared items MNEMONIC, PRINT_NAME, ALT_PRINT_NAME, DISP_NAME, SIZE, MASSBUS_FLAG, and DRIVE_TYPE. Following are explanations of each of these declared items.

MNEMONIC

The mnemonic is a two-character ASCII field that is the device mnemonic, as found in the Device Control Block (DCB). Records should be kept in alphabetical order by mnemonic.

PRINT_NAME

This six-character ASCII field identifies the particular device. This field is used in the printing of the Device Identification Information section of FULL or REGISTER reports whenever the device registers are available. In general, this field is used unless devices are being mounted or dismounted. In those cases, the device registers are not available and, depending on the device, there may be insufficient information to completely identify a device. When this occurs, the ALT_PRINT_NAME field is used instead.

ALT_PRINT_NAME

This twelve-character ASCII field identifies the device when the device registers are not available, usually for mounts and dismounts. In these cases, depending on the device, there may be insufficient information to identify a device completely. For example, when an RP04 is mounted, the only information available that can identify the device is the mnemonic DB and the device size. This information is the same for an RP04 and an RP05. In this case, the ALT_PRINT_NAME field is used, which identifies the device as an RP04/05.

DISP_NAME

This six-character ASCII field identifies the name of the device-level module used to process error-logging entries for the particular device.

SIZE

This longword specifies the number of blocks on the device. There are two special values associated with this field: a value of zero (0) indicates that the device is a magtape, and a value of -1 indicates there is no fixed size for the device. DEVSM1 will not correctly handle combinations of fixed- and variable-size devices having the same mnemonic.

MASSBUS_FLAG

This logical value indicates whether or not the device is a MASSBUS device. Set it TRUE for MASSBUS devices, and FALSE for any other devices.

DRIVE_TYPE

This byte specifies the MASSBUS drive-type value. Each MASSBUS device has a unique value which is available in the low byte of the drive-type register. If the record is not for a MASSBUS device, this field should be zero (0).

Once the record has been added to the source module (use SLP so multiple corrections can be easily merged) the DEVSM1 module must be recompiled. The first step in this process is to extract the symbol file for the DSP2M1 module (or DSP2P1 for RSX-11M-PLUS) from the ERRLOG.ULB library. The command should be:

```
> LBR DSP2M1.SYM=ERRLOG.ULB/EX:DSP2M1 (for RSX-11M) or
```

```
> LBR DSP2P1.SYM=ERRLOG.ULB/EX:DSP2P1 (for RSX-11M-PLUS)
```

Once this is done, DEVSM1 can be recompiled. The RSX-11M command sequence is:

```
> CFL
CFL> DEVSM1,DEVSM1,DEVSM1=DEVSM1,DSP2M1
Option> LITERAL SUPPORT.RSX_11M = TRUE
Option> LITERAL SUPPORT.RSX_11M PLUS = FALSE
Option> LITERAL SUPPORT.IO_ACTIVITY = FALSE
Option> /
CFL> ^Z
```

The RSX-11M-PLUS command sequence is:

```
> CFL
CFL> DEVSM1,DEVSM1,DEVSM1=DEVSM1,DSP2P1
Option> LITERAL SUPPORT.RSX_11M = FALSE
Option> LITERAL SUPPORT.RSX_11M PLUS = TRUE
Option> LITERAL SUPPORT.IO_ACTIVITY = TRUE
Option> /
CFL> ^Z
```

There is no need to recompile the DEVUDA module as no new variables are created in this process.

The updated DEVSM1 module can be replaced in the control file library with the command:

```
> LBR ERRLOG.ULB/RP=DEVSM1.ICF
```

Once this is done, the Error Logging System will be able to associate the mnemonic of the device with a module used to process entries for that device.

ERROR LOG CONTROL FILE ARCHITECTURE

At this point you should include the device-level module (and notes module, if required) in the error log library. This is done by using the command:

```
>LBR ERRLOG.ULB/IN=device_level_module.ICF[,notes_module.ICF]
```

The EUNKWN module is used (with a warning message) if an attempt is made to process an error log entry for a device that is listed in the DEVICE INFO table and whose corresponding device-level module is unavailable.

4.6 CODE EXAMPLES

The following sections consist of examples of source code from the Error Logging System. These examples are annotated for use with the preceding narrative text. They are written in the Control File Language, which is documented in the next chapter. The examples in this chapter are:

- ERM23 device-level module for RM02s and RM03s
- DSP2M1 dispatcher module for RSX-11M
- DSP2P1 dispatcher module for RSX-11M-PLUS
- NRM23 notes module for RM02s and RM03s

4.6.1 RM02/03 Device-Level Module ERM23

Following is an annotated listing of ERM23.MAC, the device-level module for the RM02 and RM03 disk drives.

①

```
MODULE ERM23 'M01.01' ;
!
! ERROR LOG CONTROL FILE MODULE: RM02, RM03
!
!           COPYRIGHT (c) 1981 BY
!   DIGITAL EQUIPMENT CORPORATION, MAYNARD
!   MASSACHUSETTS.  ALL RIGHTS RESERVED.
!
! THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
! AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE
! AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
! SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
! OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
! OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
!
! THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
! NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
! EQUIPMENT CORPORATION.
!
! DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
! ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
!
! VERSION 01.01
!
! ROBERT E. LI           08-JAN-81
```

ERROR LOG CONTROL FILE ARCHITECTURE

Device-Level Module

```

!
! This is one of the many device modules, which is called by the device
! error dispatcher (DSP2cl) or device information dispatcher (DSP3cl)
! to process all the device dependent information.
!
! Modified by:
!
!      CBP      Correct BAE/CS3 register logic

```

2

PROCEDURE DEVICE_ENTRY

```

!
! This procedure, which is called via COROUTINE statement from a dispatch
! module, declares and translates all device registers or data fields of
! the data subpacket. The intermodule variables required by the dispatch
! modules are stuffed with the appropriate values, followed by a COROUTINE
! back to the dispatch module. The dispatch module then COROUTINES back to
! this routine a second time, at a point where a write group is used to
! print the details of a FULL or REGISTER report.
!
BEGIN
!
! Declare a variable to hold the length of the subpacket.
!
DECLARE PACKET_LENGTH ;
      TEMP      :BYTE ;
END_DECLARE ;
!
! Now get the length of the DATA subpacket. Remember that the returned value
! is in bytes and includes two bytes for the length word.
!
SET PACKET_LENGTH.TEMP TO %LOK_$LENGTH(DISP.NEXT_PACKET) ;
!
! Define the data subpacket offsets and all the print information.
!

```

3

```

SUBPACKET REGISTER = DISP.NEXT_PACKET NAMED ;

```

4

```

RMCS1:      WORD MACHINE ;
      :      BIT [15]:      '*Special Condition set' ;
RMCS1_TRE:  BIT [14]:      '*Transfer Error' ;
      :      BIT [13]:      '*MASSBUS Control Bus Parity Err' ;
      :      BIT [12]:      '*Unused bit set' ;
      :      BIT [11]:      ' Drive Available',
      :                  '*Drive not Available (other port using it)' ;
      :      BIT [10]:      ' Unibus B Selected for Data Transfer',
      :                  ' Unibus A Selected for Data Transfer' ;

RMCS1_BA:   FIELD [8:2]:    ' BA17,BA16 = '
      :                  | %CNV_$BINARY(REGISTER.RMCS1_BA, 2, '0')
      :                  | ' (B)' ;
      :      BIT [ 7]:      ' Controller Ready',
      :                  ' Controller not Ready' ;
      :      BIT [ 6]:      ' Interrupt Enabled',
      :                  ' Interrupt not Enabled' ;
RMCS1_FN:   FIELD [1:5]:    ' Function = '
      :                  | INTERMOD_DEVERR.DEV_FUNCTION ;
      :      BIT [ 0]:      '*Go bit on' ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

RMWC:      WORD MACHINE ;
:          FIELD [0:16]:  %CNV_$DECIMAL_P(%COM_$NEGATE(REGISTER.RMWC), 6)
                        | ' words remaining' ;

RMBA:      WORD MACHINE ;
:          FIELD [0:16]:  ' Bus Address Register' ;

RMDA:      WORD MACHINE ;
:          FIELD [13:3]:  '*Unused bits set', NULL ;
RMDA_HD:    FIELD [8:5]:  ' Track Address = '
                        | %CNV_$DECIMAL_P(REGISTER.RMDA_HD, 2) ;
:          FIELD [5:3]:  '*Unused bits set (see note)', NULL ;
RMDA_SEC:   FIELD [0:5]:  ' Sector Address = '
                        | %CNV_$DECIMAL_P(REGISTER.RMDA_SEC, 2) ;

RMCS2:      WORD MACHINE ;
:          BIT [15]:      '*Data Late' ;
RMCS2_WC:   BIT [14]:      '*Write Check Error' ;
:          BIT [13]:      '*Parity Error' ;
:          BIT [12]:      '*Nonexistent Drive' ;
:          BIT [11]:      '*Nonexistent Memory' ;
:          BIT [10]:      '*Program Error' ;
:          BIT [ 9]:      '*Missed Transfer' ;
:          BIT [ 8]:      '*MASSBUS Data Bus Parity Error' ;
:          BIT [ 7]:      ' Output Ready (silo contains data)',
                        ' Output not Ready (silo empty)' ;
:          BIT [ 6]:      ' Input Ready (silo not full)',
                        ' Input not Ready (silo full)' ;
:          BIT [ 5]:      ' Controller Clear '
                        | '(clears all drives as well)' ;
:          BIT [ 4]:      '*Parity Test set (even parity)',
                        ' Parity Test reset (odd parity)' ;
:          BIT [ 3]:      '*Bus Address Increment Inhibit' ;
RMCS2_UN:   FIELD [0:3]:  ' Drive Selected = '
                        | INTERMOD_DEVERR.PHYS_UNIT ;

RMDS:      WORD MACHINE ;
:          BIT [15]:      ' Attention Active' ;
RMDS_ERR:   BIT [14]:      '*Error (RMER1,2 have bits set)' ;
:          BIT [13]:      ' Position in Progress' ;
:          BIT [12]:      ' Medium Online', '*Medium not Online' ;
:          BIT [11]:      ' Drive is Write Locked',
                        ' Drive is Write Enabled' ;
:          BIT [10]:      ' Last Sector Transferred (last of the pack)' ;
:          BIT [ 9]:      ' Programmable (ports program selectable)' ;
:          BIT [ 8]:      ' Drive Present', '*Drive not Present' ;
:          BIT [ 7]:      ' Drive Ready',
                        ' Drive not Ready' ;
:          BIT [ 6]:      ' Volume Valid', '*Volume not Valid' ;
:          FIELD [1:5]:  '*Unused bits set', NULL ;
:          BIT [ 0]:      ' Drive in Offset Mode',
                        ' Drive not in Offset Mode' ;

RMER1:      WORD MACHINE ;
RMER1_DCK:  BIT [15]:      '*Data Check' ;
:          BIT [14]:      '*Drive Unsafe' ;
:          BIT [13]:      '*Operation Incomplete' ;
:          BIT [12]:      '*Drive Timing Error' ;
:          BIT [11]:      '*Write Lock Error' ;
:          BIT [10]:      '*Invalid Address Error' ;
:          BIT [ 9]:      '*Address Overflow Error' ;
:          BIT [ 8]:      '*Header CRC Error' ;
:          BIT [ 7]:      '*Header Compare Error' ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

RMER1_ECH: BIT [ 6]: '*ECC Hard Error' ;
: BIT [ 5]: '*Write Clock Fail' ;
: BIT [ 4]: '*Format Error' ;
: BIT [ 3]: '*Parity Error' ;
: BIT [ 2]: '*Register Modification Refused' ;
: BIT [ 1]: '*Illegal Register' ;
: BIT [ 0]: '*Illegal Function' ;

RMAS: WORD MACHINE ;
: FIELD [8:8]: %CND $IF(REGISTER.RMDT [11],
: NULL, '*Unused bits set'),
: NULL ;
: BIT [ 7]: ' Unit #7 Attention' ;
: BIT [ 6]: ' Unit #6 Attention' ;
: BIT [ 5]: ' Unit #5 Attention' ;
: BIT [ 4]: ' Unit #4 Attention' ;
: BIT [ 3]: ' Unit #3 Attention' ;
: BIT [ 2]: ' Unit #2 Attention' ;
: BIT [ 1]: ' Unit #1 Attention' ;
: BIT [ 0]: ' Unit #0 Attention' ;

RMLA: WORD MACHINE ;
: FIELD [11:5]: '*Unused bits set', NULL ;
RMLA_ANG: FIELD [6:5]: ' Sector Count = '
: | %CNV $DECIMAL_P(REGISTER.RMLA_ANG, 2) ;
: FIELD [0:6]: '*Unused bits set', NULL ;

RMDb: WORD MACHINE ;
: FIELD [0:16]: ' Data Buffer contents' ;

RMMR1: WORD MACHINE ;
: BIT [15]: %CND $IF(REGISTER.RMMR1_MM,
: ' Debug Clock set', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Debug Clock reset', NULL) ;
: BIT [14]: %CND $IF(REGISTER.RMMR1_MM,
: ' Debug Clock Enabled', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Debug Clock Disabled', NULL) ;
: BIT [13]: %CND $IF(REGISTER.RMMR1_MM,
: ' Diagnostic End of Block set', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Diagnostic End of Block reset', NULL) ;
: BIT [12]: %CND $IF(REGISTER.RMMR1_MM,
: ' Search Time Out disabled', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Search Time Out enabled', NULL) ;
: BIT [11]: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Clock set', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Clock reset', NULL) ;
: BIT [10]: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Read Data set', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Read Data reset', NULL) ;
: BIT [ 9]: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Unit Ready', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance Unit Not Ready', NULL) ;
: BIT [ 8]: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance On Cylinder', NULL),
: %CND $IF(REGISTER.RMMR1_MM,
: ' Maintenance not On Cylinder', NULL) ;
: BIT [ 7]: %CND $IF(REGISTER.RMMR1_MM,
: '*Maintenance Seek Error', NULL) ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

:      BIT [ 6]:      %CND $IF(REGISTER.RMMR1_MM,
:                      '*Maintenance Drive Fault',NULL) ;
:      BIT [ 5]:      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Sector Pulse set', NULL),
:                      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Sector Pulse reset', NULL) ;
:      BIT [ 4]:      '*Unused bit set' ;
:      BIT [ 3]:      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Write Protect', NULL),
:                      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Write Enabled', NULL) ;
:      BIT [ 2]:      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Index Pulse set', NULL),
:                      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Index Pulse reset', NULL) ;
:      BIT [ 1]:      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Sector Compare set', NULL),
:                      %CND $IF(REGISTER.RMMR1_MM,
:                      'Maintenance Sector Compare reset', NULL);
RMMR1_MM:  BIT [ 0]:  ' Diagnostic Mode on',
:                  ' Diagnostic Mode off' ;

RMDT:      WORD MACHINE ;
:      BIT [15]:      '*Drive not Sector Addressable' ;
:      BIT [14]:      '*Unit is a Tape Drive' ;
:      BIT [13]:      NULL, '*Unit is not a Moving Head Device' ;
:      BIT [12]:      '*Unused bit set' ;
:      BIT [11]:      ' DRQ on (dual port unit)',
:                  ' DRQ off (single port unit)' ;
:      FIELD [9:2]:    '*Unused bits set', NULL ;
RMDT_TYP:  FIELD [0:8]: ' Drive Type = '
:                  | INTERMOD_DEVERR.DRIVE_TYPE ;

RMSN:      WORD MACHINE ;
:      FIELD [0:16]:   ' Drive Serial Number = '
:                  | %CNV_$BCD(REGISTER.RMSN,4) | ' (BCD)' ;

RMOF:      WORD MACHINE ;
:      FIELD [13:3]:   '*Unused bits set', NULL ;
:      BIT [12]:      ' 16 Bit Data Format',
:                  '*18 Bit Data Format' ;
:      BIT [11]:      ' ECC Inhibit', ' ECC enabled' ;
:      BIT [10]:      ' Header Compare Inhibit',
:                  ' Header Compare Enabled' ;
:      FIELD [8:2]:    '*Unused bits set', NULL ;
:      BIT [ 7]:      ' Offset Direction = Forward',
:                  ' Offset Direction = Reverse' ;
:      FIELD [0:7]:    '*Unused bits set', NULL ;

RMDC:      WORD MACHINE ;
:      FIELD [10:6]:   '*Unused bits set', NULL ;
RMDC_DC:   FIELD [0:10]: ' Desired Cylinder = '
:                  | %CNV_$DECIMAL_P(REGISTER.RMDC_DC, 4) ;

RMHR:      WORD MACHINE ;
:      FIELD [0:16]:   ' Holding Register contents' ;
RMMR2:     WORD MACHINE ;
:      BIT [15]:      ' Port A Request for Service' ;
:      BIT [14]:      ' Port B Request for Service' ;
:      BIT [13]:      ' Control Select Tag on' ;
:      BIT [12]:      %CND $IF(REGISTER.RMMR1_MM,
:                  ' Test Sequencer Branching on', NULL) ;
:      BIT [11]:      ' Control or Cylinder Tag on' ;
:      BIT [10]:      ' Control or Head Tag on' ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

RMMR2_MBL: FIELD [0:10]:  %CND $IF(REGISTER.RMMR1_MM,
                          ' Maintenance Bus Lines = '
                          | %CNV $BINARY(REGISTER.RMMR2_MBL, 10, '0')
                          | ' (B)'
                          , NULL) ;

RMER2:      WORD MACHINE ;
:          BIT [15]:      '*Bad Sector Detected (hdr bit)' ;
:          BIT [14]:      '*Seek Incomplete' ;
:          BIT [13]:      '*Operator Plug Error (removed)' ;
:          BIT [12]:      '*Invalid Command (VV bit reset)' ;
:          BIT [11]:      '*Loss of System Clock' ;
:          BIT [10]:      '*Loss of Bit Clock' ;
:          FIELD [8:2]:    '*Unused bits set', NULL ;
:          BIT [ 7]:      '*Device Check' ;
:          FIELD [4:3]:    '*Unused bits set', NULL ;
:          BIT [ 3]:      '*Data Parity Error' ;
:          FIELD [0:3]:    '*Unused bits set', NULL ;

RMEC1:      WORD MACHINE ;
:          FIELD [13:3]:    '*Unused bits set', NULL ;
RMEC1_PS:   FIELD [0:13]:   ' ECC Position = ' | VAR.ECCPS ;

RMEC2:      WORD MACHINE ;
:          FIELD [11:5]:    '*Unused bits set', NULL ;
:          FIELD [0:11]:    ' ECC Pattern = ' | VAR.ECCPAT ;

IF DEVICE_OP.FLG_BAE AND (PACKET_LENGTH.TEMP EQ #BD'46')
!
! If the RH70 flag is true and the packet length is 22 registers,
! declare the BAE and CS3 registers. Note that the packet length check
! is necessary because unmapped RSX systems will not log BAE and CS3
! even if the controller is an RH70.
!
THEN
BEGIN
RMBAE:      WORD MACHINE ;
:          FIELD [6:10]:    '*Unused bits set', NULL ;
RMBAE_EXT:  FIELD [0:6]:    ' BA21 through BA16 = '
                          | %CNV $BINARY(REGISTER.RMBAE_EXT, 6, '0') ;

RMCS3:      WORD MACHINE ;
:          BIT [15]:      '*Address Parity Error' ;
:          BIT [14]:      '*Data Parity Error, Odd Word' ;
:          BIT [13]:      '*Data Parity Error, Even Word' ;
:          BIT [12]:      '*Write Check Error, Odd Word' ;
:          BIT [11]:      '*Write Check Error, Even Word' ;
:          BIT [10]:      ' Double Word Transferred' ;
:          FIELD [7:3]:    '*Unused bits set', NULL ;
:          BIT [ 6]:      ' Interrupt Enabled',
                          ' Interrupt not Enabled' ;
:          FIELD [4:2]:    '*Unused bits set', NULL ;
RMCS3_IPC :   FIELD [0:4]:   ' Inverse Parity Check Bits = '
                          | %CNV $BINARY(REGISTER.RMCS3_IPC, 4, '0')
                          | ' (B)' ;

END ;
END_IF ;
END_PACKET ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

5

```

!
! Declare all variables needed for the subpacket print information.
!
DECLARE VAR ;
    ECCPS:          ASCII [22] ;      ! ECC position.
    ECCPAT:         ASCII [22] ;      ! ECC pattern.
END_DECLARE ;
!
! Create the device function code conversion table.
!
TABLE FUNCTION ;
    FUN_CODE:       BYTE MACHINE ;
    FUN_TEXT:       ASCII [27] ;
BEGIN TABLE
    #BO'00',        'No Operation' ;
    #BO'02',        'Seek Command' ;
    #BO'03',        'Recalibrate' ;
    #BO'04',        'Drive Clear' ;
    #BO'05',        'Release (dual port)' ;
    #BO'06',        'Offset Command' ;
    #BO'07',        'Return to Centerline' ;
    #BO'10',        'Read_in Preset' ;
    #BO'11',        'Pack Acknowledge' ;
    #BO'14',        'Search Command' ;
    #BO'24',        'Write Check Data' ;
    #BO'25',        'Write Check Header and Data' ;
    #BO'30',        'Write Data' ;
    #BO'31',        'Write Header and Data' ;
    #BO'34',        'Read Data' ;
    #BO'35',        'Read Header and Data' ;
END_TABLE ;
!
! Calculate the ECC Position.
!
! Determine if the ECC position is normal (not used), has an illegal
! value, points to the starting bit within the sector or is irrelevant.
!
IF REGISTER.RMECl_PS LE #WD'4128'
THEN
    !
    ! At this point, the ECC position is within range (0. to 4128.).
    ! Next, find out if the ECC position counter (register) was used.
    ! If the ECC position register value equals an octal 4066, it
    ! indicates the register was initialized but not used.
    !
    SET VAR.ECCPS TO %CND $IF(REGISTER.RMECl_PS EQ #WO'4066',
        'Normal', %CNV_$DECIMAL_P(REGISTER.RMECl_PS, 6)) ;
ELSE
    SET VAR.ECCPS TO 'Outside of legal range' ;
END_IF ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

!
! If the error was a non-correctable hard error or Error Correction
! was inhibited, then the ECC position and ECC pattern are irrelevant.
!
IF (REGISTER.RMER1_ECH EQ TRUE)
THEN
    BEGIN
        SET VAR.ECCPS TO 'Irrelevant (ECH set)' ;
        SET VAR.ECCPAT TO 'Irrelevant (ECH set)' ;
        END ;
    END_IF ;

IF (REGISTER.RMOF [11] EQ TRUE)
THEN
    BEGIN
        SET VAR.ECCPS TO 'Irrelevant (ECI set)' ;
        SET VAR.ECCPAT TO 'Irrelevant (ECI set)' ;
        END ;
    ELSE
        SET VAR.ECCPAT TO %CNV_$OCTAL(REGISTER.RMEC2 [0:11], 4, '0') | ' (0)' ;
    END_IF ;

```

6

```

!
! The following will use the register information to determine the
! value of the intermodule variables, which are needed by the
! dispatcher and stuff these accordingly
!
! The variables are:
!
!     INTERMOD_DEVERR.DRIVE SN
!     INTERMOD_DEVERR.DEV FUNCTION
!     INTERMOD_DEVERR.PHYS UNIT
!     INTERMOD_DEVERR.ERROR_CYLINDER
!     INTERMOD_DEVERR.ERROR_SECTOR
!     INTERMOD_DEVERR.ERROR_HEAD
!     INTERMOD_DEVERR.ERROR_GROUP      (not applicable to this device)
!     INTERMOD_DEVERR.BLOCK_NUMBER
!     INTERMOD_DEVERR.ERROR_TYPE
!
! Return the drive serial number.
!
SET INTERMOD_DEVERR.DRIVE_SN TO %CNV_$BCD(REGISTER.RMSN, 12, ' ') ;
!
! Lookup the function code in the function table.

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

!
FIND FUNCTION FUN_CODE = REGISTER.RMCS1_FN ;
!
! Check if a match is found between the register and the table,
!
IF FUNCTION.CONTEXT
THEN
!
! Yes, return the associated function text in the variable.
!
SET INTERMOD_DEVERR.DEV_FUNCTION TO FUNCTION.FUN_TEXT ;
ELSE
!
! Otherwise, return text indicating an invalid function.
!
SET INTERMOD_DEVERR.DEV_FUNCTION TO 'Invalid function' ;
END_IF ;
!
! Return the physical unit number.
!
SET INTERMOD_DEVERR.PHYS_UNIT TO %CNV_$DECIMAL(REGISTER.RMCS2_UN, 1) ;
!
! DISK GEOMETRY INFORMATION.
!
! Calculate the intermodule variables for LBN, GROUP, CYLINDER, TRACK,
! and SECTOR address, initially assuming the error packet was NOT caused
! by a data error.
!
! Calculate LBN using the formula...
!
! LBN = ( CYLINDER_ADRS * number of SECTORS/CYL +
!         HEAD_ADRS * number of SECTORS/TRACK +
!         SECTOR_ADRS )
!
SET INTERMOD_DEVERR.BLOCK_NUMBER TO
    %CNV_$DECIMAL_P(
        (REGISTER.RMDC_DC * #LD'160' +
         REGISTER.RMDA_HD * #WD'32' +
         REGISTER.RMDA_SEC ),
        9) ;
!
! Initialize GROUP. (not applicable to this device)
!
SET INTERMOD_DEVERR.ERROR_GROUP TO 'N/A' ;
!
! Initialize CYLINDER.
!
SET INTERMOD_DEVERR.ERROR_CYLINDER TO
    %CNV_$DECIMAL_P(REGISTER.RMDC_DC, 3) ;
!
! Initialize TRACK (head).
!
SET INTERMOD_DEVERR.ERROR_HEAD TO
    %CNV_$DECIMAL_P(REGISTER.RMDA_HD, 2) ;
!
! Initialize SECTOR.
!
SET INTERMOD_DEVERR.ERROR_SECTOR TO
    %CNV_$DECIMAL_P(REGISTER.RMDA_SEC, 2) ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

!
! Correct the geometry information if necessary.
!
! Upon a data error, the hardware will update the GROUP, CYLINDER, TRACK and
! SECTOR to point to the sector following the sector in error. In order to
! make the intermodule variables for GROUP, CYLINDER, TRACK, SECTOR and LBN
! point to the media address causing a data error, they are corrected (backed
! off by 1) using the following algorithm.
!
! Was it a data error ?                (check error bits)
! Yes, it was a data error.            (correction (backoff) is needed)
!   Decrement LBN.                     (recalculate pointing to previous BLK)
!   Was SECTOR = 0 ?                   (sector underflow boundry?)
!     Yes, SECTOR = 0.                  (underflow sector and borrow from TRK)
!       SECTOR = SECTORMAX.             (underflow the sector)
!       Was TRACK = 0?                  (track underflow boundry?)
!       Yes, TRACK = 0.                  (underflow TRK and borrow from CYL)
!       TRACK = TRACKMAX.               (underflow the track)
!       Decrement CYLINDER.             (borrow from CYL for TRK)
!       No, TRACK NOT = 0.              (no undeflow of TRK)
!       Decrement TRACK.                (Simply, with no borrow from CYL)
!       No, SECTOR NOT = 0.             (no underflow at all)
!       Decrement SECTOR.               (point to the previous block)
!   No, it was not a data error.        (no correction (backoff) needed)
!
! Was it a data error?
!
IF REGISTER.RMER1_DCK OR REGISTER.RMER1_ECH OR REGISTER.RMCS2_WC
THEN
!
! Yes, it was a data error. (LBN and geometry information needs correction)
!
BEGIN
!
! Correct the LBN by recalculating (backed off by one block).
!
SET INTERMOD_DEVERR.BLOCK_NUMBER TO
  %CNV_$DECIMAL P(
    (REGISTER.RMDC_DC * #LD'160' +
     REGISTER.RMDA_HD * #WD'32' +
     REGISTER.RMDA_SEC) -1,
    9) ;

!
! Was the sector address zero? (Sector underflow?)
!
IF REGISTER.RMDA_SEC EQ #BD'00'
THEN
!
! Yes, it was zero. (so undeflow the sector and borrow from track)
!
BEGIN
!
! Underflow the sector.
!
SET INTERMOD_DEVERR.ERROR_SECTOR TO '31.' ;
!
! Was track (head) address zero? (track underflow?)
!
IF REGISTER.RMDA_HD EQ #BD'00'
THEN
!
! Yes, the track was 0, so underflow the track
! and borrow from the cylinder.

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

!
BEGIN
!
! Underflow the track (head).
!
SET INTERMOD_DEVERR.ERROR_HEAD TO '4.' ;
!
! Borrow from the cylinder.
!
SET INTERMOD_DEVERR.ERROR_CYLINDER TO
    %CNV_$DECIMAL_P(REGISTER.RMDC_DC - 1, 3) ;
END ;
ELSE
!
! No, the track was not zero. Simply decrement it. (no track underflow)
!
SET INTERMOD_DEVERR.ERROR_HEAD TO
    %CNV_$DECIMAL_P(REGISTER.RMDA_HD - 1, 2) ;
END_IF ;
END ;
ELSE
!
! No, the sector address was not zero. Simply decrement it.
! (no sector underflow)
!
SET INTERMOD_DEVERR.ERROR_SECTOR TO
    %CNV_$DECIMAL_P(REGISTER.RMDA_SEC - 1, 2) ;
END_IF ;
END ;
END_IF ;

```

7

```

!
! Find the reason causing this error packet and set the variable
! accordingly.
!
IF REGISTER.RMCS1_TRE
THEN
BEGIN
    IF NOT REGISTER.RMDS_ERR
    THEN
        DECODE
            INTERMOD_DEVERR.ERROR_TYPE = REGISTER ;
            RMCS2 [15] ;      ! Data Late
            RMCS2 [14] ;      ! Write Check Error
            RMCS2 [13] ;      ! U.B. Parity Error
            RMCS2 [12] ;      ! Nonexistent Drive
            RMCS2 [11] ;      ! Nonexistent Memory
            RMCS2 [10] ;      ! Program Error
            RMCS2 [ 9] ;      ! Missed Transfer
            RMCS2 [ 8] ;      ! MASSBUS Data Bus Parity Error
        END_DECODE ;
    END ;
END ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

```

ELSE
  DECODE
    INTERMOD_DEVERR.ERROR_TYPE = REGISTER ;
    RMER2 [15] ;      ! Bad Sector Detected (Hdr bit)
    RMER2 [14] ;      ! Seek Incomplete
    RMER2 [13] ;      ! Operator Plug Error (removed)
    RMER2 [12] ;      ! Invalid Command (VV bit reset)
    RMER2 [11] ;      ! Lost of System Clock
    RMER2 [10] ;      ! Lost of Bit Clock
    RMER2 [ 7] ;      ! Device Check
    RMER2 [ 3] ;      ! Data Parity Error
    RMER1 [ 6] ;      ! ECC Hard Error
    RMER1 [15] ;      ! Data Check
    RMER1 [14] ;      ! Drive Unsafe
    RMER1 [13] ;      ! Operation Incomplete
    RMER1 [12] ;      ! Drive Timing Error
    RMER1 [11] ;      ! Write Lock Error
    RMER1 [10] ;      ! Invalid Address Error
    RMER1 [ 9] ;      ! Address Overflow Error
    RMER1 [ 8] ;      ! Header CRC Error
    RMER1 [ 7] ;      ! Header Compare Error
    RMER1 [ 5] ;      ! Write Clock Fail
    RMER1 [ 4] ;      ! Format Error
    RMER1 [ 3] ;      ! Parity Error
    RMER1 [ 2] ;      ! Register Modification Refused
    RMER1 [ 1] ;      ! Illegal Register
    RMER1 [ 0] ;      ! Illegal Function
  END_DECODE ;
END_IF ;
END ;
ELSE
  DECODE
    INTERMOD_DEVERR.ERROR_TYPE = REGISTER ;
    NOT RMDS [12] ;    ! Medium not Online
    NOT RMDS [ 8] ;    ! Drive not Present
    NOT RMDS [ 6] ;    ! Volume not Valid
    RMCS1 [13] ;      ! MASSBUS Control Bus Parity Error
    NOT RMCS1 [11] ;   ! Drive not Available
    NOT RMCS1 [ 7] ;   ! Controller not Ready
  END_DECODE ;
END_IF ;

IF (INTERMOD_DEVERR.ERROR_TYPE EQ NULL)
THEN
  SET INTERMOD_DEVERR.ERROR_TYPE TO 'No error bit found' ;
END_IF ;

```

8

```

!
! All the intermodule variables have been stuffed, so return to the
! coroutine caller (calling dispatch module).
!
COROUTINE ;
!
! The dispatcher returns control to this module here, with the flag
! INTERMOD_DEVERR.PRINT_FLAG set to either TRUE or FALSE. If the
! flag is TRUE, a FULL or REGISTER report is in progress, the banner
! has been printed, and this module prints device registers (or data
! fields for packet oriented devices) Otherwise, this module does
! not print anything, and simply exits back to the dispatcher. The
! width of the report (80/132) is controlled by dispatcher defined
! format variables REPORT.W_G_F_1 and REPORT.W_G_F_2 based on the
! user specified /WIDTH switch.
!

```


ERROR LOG CONTROL FILE ARCHITECTURE
Device-Level Module

9

```

IF INTERMOD_DEVERR.PRINT_FLAG
THEN
  BEGIN
    !
    ! Print the header for the Name, Value and Interpretation fields.
    !
    WRITE
      FORMAT
        '!5FCName!13FCValue!25FCInterpretation!2FL' ;
    !
    ! Print the registers according to the format variable (80/132)
    ! provided by the dispatcher.
    !
    WRITE_GROUP REGISTER
      FORMAT
        !
        ! Print format for the register name
        ! and it's associated value.
        !
        REPORT.W_G_F_1,
        !
        ! Print format for the exploded bits and fields.
        !
        REPORT.W_G_F_2 ;
    !
    ! If there are any NOTES to be printed, this is where the
    ! PUT of note indicies is done on the note file. When the
    ! return from this module is done, the dispatching module
    ! examines the note file to determine if the note module
    ! NRM23 should be called to print the notes specified by
    ! index number.
    !
    ! If the unused bits 5 to 7 are set in the RMDA register.
    !

```

10

```

IF (REGISTER.RMDA [5:3] NE #BB'0')
THEN
  !
  ! Print the note saying that it may cause an invalid
  ! sector address to be recognized resulting in a
  ! possible invalid address error.
  !
  PUT NOTE_NUMBERS INDEX = 1 ;
END_IF ;
END ;
END_IF ;
END ;

```

11

```

END_MODULE ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

4.6.2 DSP2M1 Dispatcher Module for RSX-11M

Following is an annotated listing of the DSP2M1 dispatcher module for RSX-11M.

```

MODULE DSP2M1 'M01.00' ;
!
! ERROR LOG CONTROL FILE MODULE: DSP2M1
!
!          COPYRIGHT (c) 1981 BY
! DIGITAL EQUIPMENT CORPORATION, MAYNARD
! MASSACHUSETTS.  ALL RIGHTS RESERVED.
!
! THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
! AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE
! AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
! SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
! OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
! OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
!
! THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
! NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
! EQUIPMENT CORPORATION.
!
! DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
! ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
!
! VERSION 01.00
!
! C. PUNTAM      22-SEP-80
!
! This module is called to process Device Error packets.
!
!   Module Name:          DSP 2 M 1
!                       --- - - -
!                       ^  ^  ^  ^
!   Module Prefix:  -----!  !  !
!                       !  !  !
!   Error Code:  -----!  !  !
!                       !  !
!   Operating System:  -----!  !
!                       !
!   Packet Format:  -----!
!
! The following Error Subcodes are defined:
!
!   Subcode      Mnemonic      Meaning
!   -----      -
!   1             E_$SDVH      Device Hard Error
!   2             E_$SDVS      Device Soft Error
!   3             E_$STMO      Device Interrupt Timeout
!
! Define any literals used in this module.
!
LITERAL DSP2 SUB ANY.FORMAT 1 =
    'I/O Operation Information:11FL' |
    '-----12FL' |
    '!5FCDevice Function!38FCType of Error!2FL' |
    '!5FC!30DP!38FC!30DP!3FL' ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

```

LITERAL DSP2 SUB ANY.FORMAT 2 =
    'Device Error Position Information:11FL' |
    '-----!2FL' |
    '!5FCCylinder!15FCGroup!22FCHead!28FCSector!36FCBlock!2FL' |
    '!5FC!8DP!15FC!5DP!22FC!4DP!28FC!6DP!36FC!10DP!2FL' ;

PROCEDURE START_MOD
BEGIN
    !
    ! Create the Subcode Conversion table.
    !
    TABLE SUBCODE ;
        NUMBER          :WORD ;
        TEXT             :ASCII [18] ;
    BEGIN_TABLE
        1,               'Device Hard Error' ;
        2,               'Device Soft Error' ;
        3,               'Device Timeout' ;
    END_TABLE ;
    !
    ! First check to see if PERIPHERAL errors are selected. If they are not,
    ! simply return. Also determine the packet subtype. If it is a known
    ! subtype code, then proceed. Otherwise it is an error.
    !
    IF NOT REPORT.PERIPHERAL
    THEN
        !
        ! This type of packet has not been selected for printing.
        !
        RETURN ;
    END_IF ;

    FIND SUBCODE NUMBER = HEADER.CODE_SUBTYPE ;

    IF NOT SUBCODE.CONTEXT
    THEN
        BEGIN
            SIGNAL 'ILLPACSB' PARAMETERS
                REPORT.PACKET_IDENT,
                %CNV_$DECIMAL(HEADER.CODE_TYPE, 3),
                %CNV_$DECIMAL(HEADER.CODE_SUBTYPE, 3) ;

            RETURN ;
        END ;
    END_IF ;

    !
    ! Find the device name by calling the DEVICE_NAME procedure.
    !
    CALL MODULE 'DEVSM1' PROCEDURE 'DEVICE_NAME' ;

    !
    ! Prepare the NOTE_NUMBERS file for any notes that may be requested.
    !
    POINTER NOTE_NUMBERS CLEAR ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

3

```
!
! Now set up the procedure DEVICE_ERROR and the appropriate device level
! module as a coroutine pair.
!
CALL MODULE INTERMOD DEVERR.DISP_NAME PROCEDURE 'DEVICE_ENTRY'
COROUTINE 'DEVICE_ERROR' ;
END ;
PROCEDURE DEVICE_ERROR
BEGIN
!
! The following is used to format the output for the
! 'Device Hard Error', 'Device Soft Error' and
! 'Device Interrupt Timeout' Device Error packets.
!
! The DEVICE_ID subpacket contains information about the
! device on which the error occurred.
!
! The DEVICE_OP subpacket contains information about the I/O
! Operation in progress on the device at the time of the error.
```

4

```
!
! Obtain information from the coroutine partner.
!
COROUTINE ;
```

5

```
!
! Assume the serial number test will succeed or be irrelevant.
!
SET INTERMOD_DEVERR.REJECT_FLAG TO FALSE ;
!
! Now test to see if this device passes the drive serial number test.
!
IF REPORT.DRIVE_SN VALID AND
  (INTERMOD_DEVERR.DRIVE_SN NE %CNV_$BCD(REPORT.DRIVE_SN, 12))
THEN
!
! Indicate that the test failed.
!
SET INTERMOD_DEVERR.REJECT_FLAG TO TRUE ;
END_IF ;
!
! Determine the type of report and format the output
! accordingly.
!
CASE REPORT.MODE OF
  ['BRIEF']:
!
! The BRIEF report is one line long.
!
BEGIN
!
! Now output the information based on the result of the test.
!
```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

6

```
IF NOT INTERMOD_DEVERR.REJECT_FLAG
THEN
    !
    ! Now output the brief report.
    !
    WRITE
        REPORT.PACKET_IDENT,
        %CNV $RSX TIME(HEADER.TIME_STAMP, 0),
        SUBCODE.TEXT,
        DISP.DEVICE_STRING,
        INTERMOD_DEVERR.ERROR_TYPE,
        'Function = ' | INTERMOD_DEVERR.DEV_FUNCTION
    FORMAT
        REPORT.BRIEF_FORMAT ;

END_IF ;
!
! Now go back to the partner. It will simply return
! without printing.
!
SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;
END ;

['FULL', 'REGISTERS']:
!
! The FULL report contains detailed information
! about the error.
!
BEGIN
!
! Now output the information based on the result of the test.
!
```

7

```
IF NOT INTERMOD_DEVERR.REJECT_FLAG
THEN
    !
    ! Output the first page if the report type is 'FULL'.
    !
    BEGIN
        IF REPORT.MODE EQ 'FULL'
        THEN
            !
            ! Now output the information for the standard subpackets.
            !
            BEGIN
                CALL MODULE REPORT.FULL_MOD_1 PROCEDURE 'OUTPUT_PACKETS' ;
                CALL MODULE REPORT.FULL_MOD_2 PROCEDURE 'OUTPUT_PACKETS' ;
                CALL MODULE REPORT.FULL_MOD_3 PROCEDURE 'OUTPUT_PACKETS' ;
                CALL MODULE REPORT.FULL_MOD_4 PROCEDURE 'OUTPUT_PACKETS' ;
```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

```

!
! Now output the Data subpacket information.
!
WRITE
    INTERMOD_DEVERR.DEV_FUNCTION,
    INTERMOD_DEVERR.ERROR_TYPE
    FORMAT
        DSP2_SUB_ANY.FORMAT_1 ;
!
! Now output the Device Error Position information
! if it is applicable.
!
IF INTERMOD_DEVERR.ERROR_CYLINDER NE '???'
THEN

    WRITE
        INTERMOD_DEVERR.ERROR_CYLINDER,
        INTERMOD_DEVERR.ERROR_GROUP,
        INTERMOD_DEVERR.ERROR_HEAD,
        INTERMOD_DEVERR.ERROR_SECTOR,
        INTERMOD_DEVERR.BLOCK_NUMBER
        FORMAT
            DSP2_SUB_ANY.FORMAT_2 ;

END_IF ;
!
! A full report is wanted, so print the record I.D.
! and header.
!
WRITE
    REPORT.PACKET_IDENT
    FORMAT
        '!1FP!5FCentry !DP!22FC(continued)!3FL' |
        'Device Supplied Information:!FL' |
        '-----!2FL' ;

END ;

ELSE
!
! Only a register dump is requested, therefore print
! the banner from the full report.
!
WRITE
    REPORT.PACKET_IDENT,
    %CNV $DECIMAL_P(HEADER.ERROR_SEQ, 8),
    DISP.DEVICE_STRING,
    SUBCODE.TEXT,
    %CND $IF((INTERMOD_DEVERR.ERROR_TYPE NE NULL),
        ' (' | INTERMOD_DEVERR.ERROR_TYPE | ')',
        NULL),
    %CNV $RSX_TIME(HEADER.TIME_STAMP, 0)

```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

```

        FORMAT
        !
        ! Select the format statement
        ! based on the desired width.
        !
        %CND_$IF(REPORT.WIDE,
        !
        ! WIDE is selected.
        !
        '!1FP!5FCentry !7DP!20FCSequence !9DP' |
        '!40FC!6DP!48FC!18DP!DP!2FS!20DP!3FL' |
        'Device Supplied Information:!FL' |
        '-----!2FL',
        !
        ! NARROW is selected.
        !
        '!1FP!5FCentry !7DP!20FCSequence !9DP!40FC!6DP!FL' |
        '!5FC!18DP!DP!2FS!20DP!3FL' |
        'Device Supplied Information:!FL' |
        '-----!2FL') ;

    END_IF ;
    !
    ! Now indicate we want to have the device module print.
    !
    SET INTERMOD_DEVERR.PRINT_FLAG TO TRUE ;
    END ;

ELSE
    !
    ! The packet was rejected. Don't print anything.
    !
    SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;

    END_IF ;
    !
    ! Now go back to the partner. It will output the device registers
    ! if the print flag is true.
    !
    END ;

        8

['NONE']:
    !
    ! If the report type is NONE, output nothing.
    !
    SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;

END_CASE ;

        9

    !
    ! Now COROUTINE back to the partner. It will print if instructed to do so.
    !
    COROUTINE ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M Dispatcher Module

⑩

```
!  
! Now see if any notes were requested and print them if there were.  
!  
IF NOTE_NUMBERS.CONTEXT THEN  
    BEGIN  
    SET INTERMOD_DEVERR.DISP_NAME TO 'N' |  
        %STR_$REMAINING(INTERMOD_DEVERR.DISP_NAME, 2) ;  
    IF %PKT_$MODULE(INTERMOD_DEVERR.DISP_NAME)  
    THEN  
        CALL MODULE INTERMOD_DEVERR.DISP_NAME PROCEDURE 'NOTES' ;  
    ELSE  
        SIGNAL 'NONOTES' PARAMETERS INTERMOD_DEVERR.DRIVE_TYPE ;  
    END_IF ;  
    END ;  
END_IF ;  
END ;  
END_MODULE ;    ! DSP2M1.CNF
```


ERROR LOG CONTROL FILE ARCHITECTURE RSX-11M-PLUS Dispatcher Module

4.6.3 DSP2P1 Dispatcher Module for RSX-11M-PLUS

Following is an annotated listing of the DSP2P1 dispatcher module for RSX-11M-PLUS.

```

MODULE DSP2P1 'P01.00' ;
!
! ERROR LOG CONTROL FILE MODULE: DSP2P1
!
! COPYRIGHT (c) 1981 BY
! DIGITAL EQUIPMENT CORPORATION, MAYNARD
! MASSACHUSETTS. ALL RIGHTS RESERVED.
!
! THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
! AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE
! AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
! SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
! OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
! OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
!
! THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
! NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
! EQUIPMENT CORPORATION.
!
! DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
! ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
!
! VERSION 01.00
!
! C. PUNTAM 22-SEP-80
!
! This module is called to process Device Error packets.
!
! Module Name: DSP 2 P 1
!             --- ^ ^ ^ ^
!
! Module Prefix: -----! !!
!
! Error Code: -----! !!
!
! Operating System: -----!
!
! Packet Format: -----!
!
! The following Error Subcodes are defined:
!
! Subcode Mnemonic Meaning
! -----
!
! 1 E_$SDVH Device Hard Error
! 2 E_$SDVS Device Soft Error
! 3 E_$STMO Device Interrupt Timeout
! 4 E_$SUNS Unsolicited Interrupt
!
! Define any literals used in this module.
!
! LITERAL DSP2_SUB_ANY.FORMAT 1 =
! 'I/O Operation Information:11FL' |
! '-----!2FL' |
! '15FCDevice Function!38FCType of Error!2FL' |
! '15FC!30DP!38FC!30DP!3FL' ;

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```

LITERAL DSP2_SUB_ANY.FORMAT_2 =
  'Device Error Position Information:11FL' |
  '-----!2FL' |
  '15FCylinder!15FCGroup!22FCHead!28FCSector!36FCBlock!2FL' |
  '15FC!8DP!15FC!5DP!22FC!4DP!28FC!6DP!36FC!10DP!2FL' ;

PROCEDURE START_MOD
BEGIN
!
! Create the Subcode Conversion table.
!
TABLE SUBCODE ;
  NUMBER          :WORD ;
  TEXT            :ASCII [18] ;
BEGIN_TABLE
  1,              'Device Hard Error' ;
  2,              'Device Soft Error' ;
  3,              'Device Timeout' ;
  4,              'Spurious Interrupt' ;
END_TABLE ;

1

!
! Create a flag that indicates whether the device is a magtape or not.
!
DECLARE INDICATE ;
  TAPE_FLAG       :LOGICAL ;
END_DECLARE ;

!
! First check to see if PERIPHERAL errors are selected. If they are not,
! simply return. Also determine the packet subtype. If it is a known
! subtype code, then proceed. Otherwise it is an error.
!
IF NOT REPORT.PERIPHERAL
THEN
!
! This type of packet has not been selected for printing.
!
RETURN ;

END_IF ;

FIND SUBCODE NUMBER = HEADER.CODE_SUBTYPE ;

IF NOT SUBCODE.CONTEXT
THEN
  BEGIN
    SIGNAL 'ILLPACSB' PARAMETERS
      REPORT.PACKET IDENT,
      %CNV_$DECIMAL(HEADER.CODE_TYPE, 3),
      %CNV_$DECIMAL(HEADER.CODE_SUBTYPE, 3) ;

    RETURN ;
  END ;
END_IF ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```

!
! Now find the device name by calling the DEVICE_NAME procedure.
!
CALL MODULE 'DEVSM1' PROCEDURE 'DEVICE_NAME' ;
!
! Prepare the NOTE_NUMBERS file for any notes that may be requested.
!
POINTER NOTE_NUMBERS CLEAR ;
!
! Now set up the procedure DEVICE_ERROR and the appropriate device level
! module as a coroutine pair.
!
CALL MODULE INTERMOD DEVERR.DISP_NAME PROCEDURE 'DEVICE_ENTRY'
      COROUTINE 'DEVICE_ERROR' ;
END ;
PROCEDURE DEVICE_ERROR
BEGIN
!
! The following is used to format the output for the
! 'Device Hard Error', 'Device Soft Error', 'Device Interrupt Timeout'
! and 'Spurious Interrupt' Device Error packets.
!
! The DEVICE_ID subpacket contains information about the
! device on which the error occurred.
!
! The I/O ACTIVITY subpacket contains information about all
! other concurrent I/O in the system.
!
! The DEVICE_OP subpacket contains information about the I/O
! Operation in progress on the device at the time of the error.
!
! Obtain information from the coroutine partner.
!
COROUTINE ;
!
! Assume the serial number test will succeed or be irrelevant.
!
SET INTERMOD_DEVERR.REJECT_FLAG TO FALSE ;
!
! Now test to see if this device passes the drive serial number test.
!
IF REPORT.DRIVE_SN_VALID AND
   (INTERMOD_DEVERR.DRIVE_SN NE %CNV_$BCD(REPORT.DRIVE_SN, 12))
THEN
!
! Indicate that the test failed.
!
SET INTERMOD_DEVERR.REJECT_FLAG TO TRUE ;
END_IF ;
!
! Determine the type of report and format the output
! accordingly.
!
CASE REPORT.MODE OF
    ['BRIEF']:
!
! The BRIEF report is one line long.
!
BEGIN
!
! Now output the information based on the result of the test.
!

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```
IF NOT INTERMOD_DEVERR.REJECT_FLAG
THEN
!
! Now output the brief report.
!
BEGIN
WRITE
    REPORT.PACKET IDENT,
    %CNV_SRSX_TIME(HEADER.TIME_STAMP, 0),
    SUBCODE.TEXT,
    DISP.DEVICE STRING,
    INTERMOD_DEVERR.ERROR_TYPE,
    'Function = ' | INTERMOD_DEVERR.DEV_FUNCTION
FORMAT
    REPORT.BRIEF_FORMAT ;

    2

!
! Now increment the printed packet count.
!
INCREMENT REPORT.PRINT_COUNT ;
END ;

END_IF ;
!
! Now go back to the partner. It will simply return
! without printing.
!
SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;
END ;

['FULL', 'REGISTERS']:
!
! The FULL report contains detailed information
! about the error.
!
BEGIN
!
! Now output the information based on the result of the test.
!
IF NOT INTERMOD_DEVERR.REJECT_FLAG
THEN
!
! Output the first page if the report type is 'FULL'.
!
BEGIN
IF REPORT.MODE EQ 'FULL'
THEN
!
! Now output the information for the standard subpackets.
!
BEGIN
CALL MODULE REPORT.FULL_MOD PROCEDURE 'OUTPUT_PACKETS' ;
!
! Now output the Data subpacket information.
!
WRITE
    INTERMOD_DEVERR.DEV_FUNCTION,
    INTERMOD_DEVERR.ERROR_TYPE
FORMAT
    DSP2_SUB_ANY.FORMAT_1 ;
```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```
!
! Now output the Device Error Position information
! if it is applicable.
!
IF INTERMOD_DEVERR.ERROR_CYLINDER NE '???'
THEN
    WRITE
        INTERMOD_DEVERR.ERROR_CYLINDER,
        INTERMOD_DEVERR.ERROR_GROUP,
        INTERMOD_DEVERR.ERROR_HEAD,
        INTERMOD_DEVERR.ERROR_SECTOR,
        INTERMOD_DEVERR.BLOCK_NUMBER
    FORMAT
        DSP2_SUB_ANY.FORMAT_2 ;

END_IF ;
!
! A full report is wanted, so print the record I.D.
! and header.
!
WRITE
    REPORT.PACKET_IDENT
    FORMAT
        '11FP!5FCentry !DP!22FC(continued)!3FL' |
        'Device Supplied Information:!FL' |
        '-----!2FL' ;

END ;

ELSE
!
! Only a register dump is requested, therefore print
! the banner from the full report.
!
WRITE
    REPORT.PACKET_IDENT,
    %CNV_$DECIMAL_P(HEADER.ERROR_SEQ, 8),
    DISP.DEVICE_STRING,
    SUBCODE.TEXT,
    %CND_$IF((INTERMOD_DEVERR.ERROR_TYPE NE NULL),
        ' (' | INTERMOD_DEVERR.ERROR_TYPE | ')',
        NULL),
    %CNV_$RSX_TIME(HEADER.TIME_STAMP, 0)
```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```

        FORMAT
        !
        ! Select the format statement
        ! based on the desired width.
        !
        %CND_$IF(REPORT.WIDE,
        !
        ! WIDE is selected.
        !
        '11FP!5FCEntry !7DP!20FCSequence !9DP' |
        '!40FC!6DP!48FC!18DP!DP!2FS!20DP!3FL' |
        'Device Supplied Information:!!FL' |
        '-----!2FL',
        !
        ! NARROW is selected.
        !
        '11FP!5FCEntry !7DP!20FCSequence !9DP!40FC!6DP!FL' |
        '!5FC!18DP!DP!2FS!20DP!3FL' |
        'Device Supplied Information:!!FL' |
        '-----!2FL') ;

    END_IF ;
    !
    ! Now increment the printed packet count and tell the device
    ! module we want it to print.
    !
    INCREMENT REPORT.PRINT_COUNT ;

    SET INTERMOD_DEVERR.PRINT_FLAG TO TRUE ;
    END ;

ELSE
    !
    ! We don't want to print because the packet was rejected.
    !
    SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;

    END_IF ;
    !
    ! Now go back to the partner. It will output the device registers
    ! if the print flag is true.
    !
    END ;

['NONE']:
    !
    ! If the report type is NONE, output nothing.
    !
    SET INTERMOD_DEVERR.PRINT_FLAG TO FALSE ;

END_CASE ;
!
! Now COROUTINE back to the partner. It will print if instructed to do so.
!
COROUTINE ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

3

```
!
! Test to see if the packet was accepted. If it was, update the files.
!
IF NOT INTERMOD_DEVERR.REJECT_FLAG THEN
    !
    ! Update the files.
    !
    CALL PROCEDURE 'UPDATE_RECORD' ;
END_IF ;
!
! Now see if any notes were requested and print them if there were.
!
IF NOTE_NUMBERS.CONTEXT THEN
    BEGIN
    SET INTERMOD_DEVERR.DISP_NAME TO 'N' |
        %STR_$REMAINING(INTERMOD_DEVERR.DISP_NAME, 2) ;
    IF %PKT_$MODULE(INTERMOD_DEVERR.DISP_NAME)
    THEN
        CALL MODULE INTERMOD_DEVERR.DISP_NAME PROCEDURE 'NOTES' ;
    ELSE
        SIGNAL 'NONOTES' PARAMETERS INTERMOD_DEVERR.DRIVE_TYPE ;
    END_IF ;
    END ;
END_IF ;
END ;
PROCEDURE UPDATE_RECORD
BEGIN
! This procedure is used to update an error type record if a record for the
! type of error exists. If it does not exist, a record is created.
!
! First see if a record exists in the ERROR_INFO_E file that matches
! on the following keys:
!
!     Device name
!     Device type
!     Pack SN
!     Drive SN
!     Volume label
!     Error type
!
```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

4

```
IF REPORT.ERROR
THEN
!
! This type of summary is desired.
!
BEGIN
POINTER ERROR_INFO_E FIRST ;
!
! Now try to find a record that matches on all of the keys.
!
FIND ERROR INFO E
  NAME = DISP.DEVICE_STRING,
  DEVICE_TYPE = %CND_$IF(INTERMOD DEVERR.MISMATCH_FLAG, '*', NULL) |
  INTERMOD DEVERR.DRIVE_TYPE,
  PACK_SN = DEVICE_ID.PACK_SN,
  DRIVE_SN = INTERMOD DEVERR.DRIVE_SN,
  VOLUME_LABEL = DEVICE_ID.VOLUME_LABEL,
  ERROR_TYPE = INTERMOD DEVERR.ERROR_TYPE ;
!
! See if there was a match.
!
IF ERROR_INFO_E.CONTEXT
THEN
!
! There was a match. Update the record to
! show that this error occurred.
!
BEGIN
INCREMENT ERROR_INFO_E.ERROR_COUNT ;

IF DISP.PACKET_DATE LT ERROR_INFO_E.FIRST_DATE
THEN
  BEGIN
    SET ERROR_INFO_E.FIRST_DATE TO DISP.PACKET_DATE ;

    SET ERROR_INFO_E.FIRST_PACKET TO REPORT.PACKET_IDENT ;
  END ;

END_IF ;

IF DISP.PACKET_DATE GT ERROR_INFO_E.LAST_DATE
THEN
  BEGIN
    SET ERROR_INFO_E.LAST_DATE TO DISP.PACKET_DATE ;

    SET ERROR_INFO_E.LAST_PACKET TO REPORT.PACKET_IDENT ;
  END ;

END_IF ;
END ;
```


ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```

ELSE
!
! This is the first error of this kind. Create a record in the
! ERROR_INFO_E file that describes this error.
!
PUT ERROR_INFO E
  NAME = DISP.DEVICE_STRING,
  DEVICE_TYPE = %CND_$IF(INTERMOD_DEVERR.MISMATCH_FLAG, '*', NULL) |
                INTERMOD_DEVERR.DRIVE_TYPE,
  PACK_SN = DEVICE_ID.PACK_SN,
  DRIVE_SN = INTERMOD_DEVERR.DRIVE_SN,
  VOLUME_LABEL = DEVICE_ID.VOLUME_LABEL,
  ERROR_TYPE = INTERMOD_DEVERR.ERROR_TYPE,
  ERROR_COUNT = 1,
  FIRST_DATE = DISP.PACKET_DATE,
  LAST_DATE = DISP.PACKET_DATE,
  FIRST_PACKET = REPORT.PACKET_IDENT,
  LAST_PACKET = REPORT.PACKET_IDENT ;

END_IF ;
END ;

END_IF ;
!
! First see if a record exists in the ERROR_INFO_G file that matches
! on the following keys:
!
!   Device name
!   Device type
!   Pack SN
!   Drive SN
!   Volume label
!   Block number
!

5

IF REPORT.GEOMETRY AND NOT INDICATE.TAPE_FLAG
THEN
!
! This type of summary is desired.
!
BEGIN
  POINTER ERROR_INFO_G FIRST ;
!
! Now try to find a record that matches on all of the keys.
!
FIND ERROR_INFO G
  NAME = DISP.DEVICE_STRING,
  DEVICE_TYPE = %CND_$IF(INTERMOD_DEVERR.MISMATCH_FLAG, '*', NULL) |
                INTERMOD_DEVERR.DRIVE_TYPE,
  PACK_SN = DEVICE_ID.PACK_SN,
  DRIVE_SN = INTERMOD_DEVERR.DRIVE_SN,
  VOLUME_LABEL = DEVICE_ID.VOLUME_LABEL,
  BLOCK_NUMBER = INTERMOD_DEVERR.BLOCK_NUMBER ;
!
! See if there was a match.
!
IF ERROR_INFO_G.CONTEXT

```

ERROR LOG CONTROL FILE ARCHITECTURE
RSX-11M-PLUS Dispatcher Module

```
THEN
!
! There was a match. Update the record to
! show that this error occurred.
!
INCREMENT ERROR_INFO_G.ERROR_COUNT ;

ELSE
!
! This is the first error of this kind. Create a record in the
! ERROR_INFO_G file that describes this error.
!
PUT ERROR_INFO_G
  NAME = DISP.DEVICE STRING,
  DEVICE_TYPE = %CND_$IF(INTERMOD_DEVERR.MISMATCH_FLAG, '*', NULL) |
               INTERMOD_DEVERR.DRIVE_TYPE,
  PACK SN = DEVICE ID.PACK SN,
  DRIVE SN = INTERMOD_DEVERR.DRIVE SN,
  VOLUME LABEL = DEVICE ID.VOLUME LABEL,
  ERROR_HEAD = INTERMOD_DEVERR.ERROR HEAD,
  ERROR_GROUP = INTERMOD_DEVERR.ERROR GROUP,
  ERROR_CYLINDER = INTERMOD_DEVERR.ERROR CYLINDER,
  ERROR_SECTOR = INTERMOD_DEVERR.ERROR SECTOR,
  BLOCK_NUMBER = INTERMOD_DEVERR.BLOCK_NUMBER,
  ERROR_COUNT = 1 ;

END_IF ;
END ;

END_IF ;
END ;

END_MODULE ;    ! DSP2P1.CNF
```


ERROR LOG CONTROL FILE ARCHITECTURE
Notes Module

4.6.4 RM02/03 Notes Module NRM23

Following is an annotated listing of the notes module for the RM02 and RM03 disk drives.

①

```
MODULE NRM23 'M01.00' ;
!
! ERROR LOG CONTROL FILE MODULE:  RM02, RM03 Notes
!
!           COPYRIGHT (c) 1981 BY
!           DIGITAL EQUIPMENT CORPORATION, MAYNARD
!           MASSACHUSETTS.  ALL RIGHTS RESERVED.
!
! THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
! AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE
! AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
! SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
! OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
! OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
!
! THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
! NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
! EQUIPMENT CORPORATION.
!
! DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
! ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
!
! VERSION 01.00
!
! R. Ryan      30-Jun-81
!
! This is one of the many device modules, which is called by the device
! error dispatcher (DSP2cl) or device information dispatcher (DSP3cl)
! to process notes for all the device dependent information.
```

②

```
PROCEDURE NOTES
!
! This procedure, which is called from the DSP2cl module, processes any
! requests for notes.
!
BEGIN
!
! Print the NOTE header and define the format for the NOTE section.
!
```

③

```
WRITE FORMAT
    '!3FLNotes on RM02, RM03 errors:!2FL' ;

POINTER NOTE_NUMBERS FIRST ;

WHILE NOTE_NUMBERS.CONTEXT DO
```

ERROR LOG CONTROL FILE ARCHITECTURE
Notes Module

4

```
BEGIN
SELECT NOTE_NUMBERS.INDEX OF
!
! Note number 1.
!
BEGIN
WRITE FORMAT
    '*      RMDA bits 5,6,7 are unused, however if they are' ;
WRITE FORMAT
    '      set, they will be interpreted as the high order' ;
WRITE FORMAT
    '      bits of the sector address. This may result in' ;
WRITE FORMAT
    '      an Invalid Address Error.!3FL' ;
END ;
ELSE
```

5

```
!
! This is an unknown note number.
!
SIGNAL 'UNKNWNNOT' PARAMETERS
    %CNV_$DECIMAL_P(NOTE_NUMBERS.INDEX, 3),
    INTERMOD_DEVERR.DRIVE_TYPE ;

END_SELECT ;
```

6

```
POINTER NOTE_NUMBERS NEXT ;
END ;

END ;
```

7

```
END_MODULE ;    ! NRM23.CNF
```


ERROR LOG CONTROL FILE ARCHITECTURE Notes Module

4.6.5 Subpacket Definitions

The following sections list the DIGITAL-standard subpackets. They are listed under the modules that declare them.

4.6.5.1 Subpackets Declared by DISPATCH - The HEADER subpacket contains information largely from SYSCM. It describes the characteristics of the system and packet. This subpacket is always required.

```

SUBPACKET HEADER = DISP.NEXT PACKET ;
  SUBPKT_FLAGS      :WORD ;                      ! E$HSBF
    SFLG_HDR        :BIT [0] ;                    ! Header Subpacket
    SFLG_TSK        :BIT [1] ;                    ! Task Subpacket
    SFLG_DID        :BIT [2] ;                    ! Device Id. Subpacket
    SFLG_DOP        :BIT [3] ;                    ! Device Op. Subpacket
    SFLG_DAC        :BIT [4] ;                    ! Device Ac. Subpacket
    SFLG_DAT        :BIT [5] ;                    ! Data Subpacket
    SFLG_MBC        :BIT [13] ;                   ! 22-bit Massabus Controller
    SFLG_CMD        :BIT [14] ;                   ! Command Subpacket
    SFLG_ZER        :BIT [15] ;                   ! I/O counts zeroed
  OP_SYS            :BYTE ;                      ! E$HSYS
  FORMAT_ID         :BYTE ;                      ! E$HIDN
  OP_SYS_ID         :ASCII [4] ;                 ! E$HSID
  CONTEXT_CODE      :BYTE ;                      ! E$HCTX
    CC_NOR          :BIT [0] ;                    ! Normal Entry
    CC_STA          :BIT [1] ;                    ! Start Entry
    CC_CDA          :BIT [2] ;                    ! CDA Entry
  FLAGS            :BYTE ;                      ! E$HFLG
    FLG_ADR         :FIELD [0:2] ;                 ! Addressing mode
    FLG_COU         :BIT [2] ;                    ! Error Counts supplied
    FLG_QBS         :BIT [3] ;                    ! Q-BUS system
  ENTRY_SEQ        :WORD ;                      ! E$HENS
  ERROR_SEQ        :WORD ;                      ! E$HERS
  CODE_TYPE        :BYTE ;                      ! E$HTYC
  CODE_SUBTYPE     :BYTE ;                      ! E$HTYS
  TIME_STAMP       :RSX_TIME ;                  ! E$HTIM
  PROC_TYPE        :BYTE ;                      ! E$HPTY
  RESERVED         :BYTE ;                      ! Reserved byte
  PROC_ID          :WORD ;                      ! E$HURM
    URM_CPU         :FIELD [0:4] ;                 ! Processor Identifier
END_PACKET ;

```

The TASK subpacket contains information either about the task that logged the packet, or the task that caused the packet to be logged.

```

SUBPACKET TASK = DISP.NEXT PACKET ;
  TASK_NAME        :LONGWORD ;                  ! E$TTSK
  UIC              :WORD ;                      ! E$TUIC
    UIC_MEMBER      :FIELD [0:8] ;                 ! Member number in UIC
    UIC_GROUP       :FIELD [8:8] ;                 ! Group number in UIC
  TI_DEV          :ASCII [2] ;                  ! E$TTID
  TI_UNIT         :BYTE ;                      ! E$TTIU
  FLGS            :BYTE ;                      ! E$TFLG
    FLG_PRV        :BIT [0] ;                    ! Privileged Task
    FLG_PRI        :BIT [1] ;                    ! Privileged Terminal

```

The DEVICE_ID subpacket contains information about the device on which the error occurred.

ERROR LOG CONTROL FILE ARCHITECTURE Notes Module

```

SUBPACKET DEVICE_ID = DISP.NEXT_PACKET ;
  MNEMONIC           :ASCII [2] ;           ! E$ILDV
  LOGICAL UNIT       :BYTE ;                 ! E$ILUN
  CONTROLLER_NUM     :BYTE ;                 ! E$IPCO
  PHYS_UNIT          :BYTE ;                 ! E$IPUN
  PHYS_SUB_UNIT      :BYTE ;                 ! E$IPSU

$IF SUPPORT.RSX_11M_PLUS

$THEN

  IF OP_SYS.SUFFIX EQ 'P'

    THEN

      BEGIN
        PHYS_DEV_MNEMON :ASCII [2] ;           ! E$IPDV
      END ;

      END_IF ;

    $END_IF

    DEV_FLAGS          :BYTE ;                 ! E$IFLG
      DFLG_SUB         :BIT [0] ;             ! Subcontroller device

    $IF SUPPORT.RSX_11M_PLUS

    $THEN

      DFLG_NUX         :BIT [1] ;             ! No UCB extension

    $END_IF

    RESERVED          :BYTE ;                 ! Reserved byte
    VOLUME LABEL      :ASCII [12] ;
    PACK_SN           :LONGWORD ;
    DEV_TYPE_CLASS    :WORD ;
    DEV_TYPE           :LONGWORD ;
    IO_COUNT          :LONGWORD ;
    SOFT_ERCNT        :BYTE ;
    HARD_ERCNT        :BYTE ;

    $IF SUPPORT.RSX_11M_PLUS

    $THEN

      IF OP_SYS.SUFFIX EQ 'P'

        THEN

          BEGIN
            WRD_XFR_COUNT :LONGWORD ;           ! E$IBLK
            CYL_CRS_COUNT :LONGWORD ;           ! E$ICYL
          END ;

          END_IF ;

        $END_IF

      END_PACKET ;

```


ERROR LOG CONTROL FILE ARCHITECTURE Notes Module

The DEVICE_OP subpacket contains information about the requested I/O operation.

```

SUBPACKET DEVICE_OP = DISP.NEXT_PACKET ;
TASK_NAME          :LONGWORD ;                ! E$OTSK
UIC                :WORD ;                    ! E$OUIC
    UIC_MEMBER      :FIELD [0:8] ;              ! Member number in UIC
    UIC_GROUP        :FIELD [8:8] ;              ! Group number in UIC
TI_DEV             :ASCII [2] ;                ! E$OTID
TI_UNIT            :BYTE ;                    ! E$OTIU
RESERVED           :BYTE ;                    ! Reserved Byte
IO_FUNCTION         :WORD ;                    ! E$OFNC
    SF_IQX           :BIT [0] ;                  ! IQ.X subfunction bit
    SF_IQQ           :BIT [1] ;                  ! IQ.Q subfunction bit
    SF_IQUMD         :BIT [2] ;                  ! IQ.UMD subfunction bit
FLAGS              :BYTE ;                    ! E$OFLG
    FLG_TRA          :BIT [0] ;                  ! Transfer operation
    FLG_DMA          :BIT [1] ;                  ! DMA device
    FLG_BAE          :BIT [2] ;                  ! 22 bit addressing device
RESERVED           :BYTE ;                    ! Reserved Byte
XFER_ADDRESS_1     :WORD ;                    ! E$OADD + 0
    XFER1_HIGH_2     :FIELD [4:2] ;              ! High Order 2 bits of address
    XFER1_HIGH_6     :FIELD [0:6] ;              ! High Order 6 bits of address
XFER_ADDRESS_2     :WORD ;                    ! E$OADD + 2
    XFER2_TAUB       :FIELD [0:13] ;             ! T. A. in units of bytes
XFER_BYTE_COUNT    :WORD ;                    ! E$OSIZ
RETRIES_LEFT       :BYTE ;                    ! E$ORTY
MAX_RETRIES        :BYTE ;                    ! E$ORTY+1
END_PACKET ;

```

The IO_ACTIVITY subpacket contains information about other I/O going on in the system at the time the error was detected.

```

SUBPACKET IO_ACTIVITY = DISP.NEXT_PACKET REPEATED ;
MNEMONIC           :ASCII [2] ;                ! E$ALDV
LOGICAL_UNIT        :BYTE ;                    ! E$ALUN
CONTROLLER_NUM      :BYTE ;                    ! E$APCO
PHYS_UNIT           :BYTE ;                    ! E$APUN
PHYS_SUB_UNIT       :BYTE ;                    ! E$APSU

$IF SUPPORT.RSX_11M_PLUS

$THEN

    IF OP_SYS.SUFFIX EQ 'P'

        THEN

            BEGIN
            PHYS_DEV_MNEMON :ASCII [2] ;          ! E$IPDV
            END ;

            END_IF ;

        $END_IF

        DEV_FLAGS         :BYTE ;                ! E$IFLG
        DFLG_SUB          :BIT [0] ;             ! Subcontroller device

```

ERROR LOG CONTROL FILE ARCHITECTURE Notes Module

```

$IF SUPPORT.RSX_11M_PLUS
$THEN
    DFLG_NUX          :BIT [1] ;                ! No UCB extension

$END_IF

TI_UNIT              :BYTE ;                    ! E$ATIU
TASK_NAME             :LONGWORD ;               ! E$ATSK
UIC                   :WORD ;                   ! E$AUIC
    UIC_MEMBER        :FIELD [0:8] ;             ! Member number in UIC
    UIC_GROUP         :FIELD [8:8] ;             ! Group number in UIC
TI_DEV                :ASCII [2] ;              ! E$ATID
IO_FUNCTION           :WORD ;                   ! E$AFNC
    SF_IQX             :BIT [0] ;                ! IQ.X subfunction bit
    SF_IQQ             :BIT [1] ;                ! IQ.Q subfunction bit
    SF_IQUMD           :BIT [2] ;                ! IQ.UMD subfunction bit
FLAGS                 :BYTE ;                   ! E$AFLG
    FLG_TRA            :BIT [0] ;                ! Transfer operation
    FLG_DMA            :BIT [1] ;                ! DMA device
    FLG_BAE            :BIT [2] ;                ! 22 bit addressing device
RESERVED              :BYTE ;                   ! Reserved Byte
XFER_ADDRESS_1        :WORD ;                   ! E$AADD + 0
    XFER1_HIGH_2       :FIELD [4:2] ;             ! High Order 2 bits of address
    XFER1_HIGH_6       :FIELD [0:6] ;             ! High Order 6 bits of address
XFER_ADDRESS_2        :WORD ;                   ! E$AADD + 2
    XFER2_TAUB         :FIELD [0:13] ;           ! T. A. in units of bytes
XFER_BYTE_COUNT       :WORD ;                   ! E$ASIZ
END_PACKET ;

```

4.6.5.2 Subpackets Declared by DSP1M1/DSP1P1 - There are several different DATA subpackets declared by DSP1M1/DSP1P1. Here are descriptions of each of them.

The following DATA subpacket (Code = 1, Subcode = 1) contains information about a "Status change" operation:

```

SUBPACKET DATA = DISP.NEXT PACKET ;
LIMIT_CODE           :BYTE ;
LOG_CODE             :BYTE ;
FLAGS                :BYTE ;
    FLG_CRE           :BIT [0] ;
FILE_SPEC_LEN        :BYTE ;
FILE_SPEC            :ASCII [80] ;
END_PACKET ;

```

The following DATA subpacket (Code = 1, Subcode = 2) contains information about a "Switch Logging Files" operation:

```

SUBPACKET DATA = DISP.NEXT PACKET ;
RESERVED             :WORD ;
FLAGS                :BYTE ;
    FLG_CRE           :BIT [0] ;
    FLG_DEL           :BIT [1] ;
FILE_SPEC_LEN        :BYTE ;
FILE_SPEC            :ASCII [80] ;
END_PACKET ;

```


ERROR LOG CONTROL FILE ARCHITECTURE
Notes Module

The following DATA subpacket (Code = 1, Subcode = 3) contains information about an "Append to File" operation:

```
SUBPACKET DATA = DISP.NEXT_PACKET ;
  RESERVED          :WORD ;
  FLAGS             :BYTE ;
    FLG_CRE         :BIT [0] ;
    FLG_DEL         :BIT [1] ;
  FILE_SPEC_LEN     :BYTE ;
  FILE_SPEC         :ASCII [80] ;
END_PACKET ;
```

The following DATA subpacket (Code = 1, Subcode = 4) contains information about a "Set Backup File" operation:

```
SUBPACKET DATA = DISP.NEXT_PACKET ;
  RESERVED          :WORD ;
  FLAGS             :BYTE ;
  FILE_SPEC_LEN     :BYTE ;
  FILE_SPEC         :ASCII [80] ;
END_PACKET ;
```

The following DATA subpacket (Code = 1, Subcode = 6) contains information about a "Change Limits" operation:

```
SUBPACKET DATA = DISP.NEXT_PACKET REPEATED ;
  HARD LIM FLAG     :BYTE ;
    NEW LIMH        :BIT [0] ;
  HARD LIMIT        :BYTE ;
  SOFT LIM FLAG     :BYTE ;
    NEW LIMS        :BIT [0] ;
  SOFT LIMIT        :BYTE ;
  MNEMONIC          :ASCII [2] ;
  LOGICAL UNIT      :BYTE ;
  RESERVED          :BYTE ;
END_PACKET ;
```

4.6.5.3 Subpackets Declared by DSP2M1/DSP2P1 - The DATA subpackets for device errors (Code = 2, Subcodes = 1, 2, 3) contain information that is specific to each device. Please see the appropriate device-level module for the format of the DATA subpacket.

4.6.5.4 Subpackets Declared by DSP3M1/DSP3P1 - The DATA subpackets for device information messages (Code = 3, Subcode = 1) contain information that is specific to each device. Please see the appropriate device-level module for the format of the DATA subpacket.

ERROR LOG CONTROL FILE ARCHITECTURE Notes Module

4.6.5.5 Subpackets Declared by DSP4M1/DSP4P1 - There is no DATA subpacket for "Mount", "Dismount", and "Device Info Reset" events (Code = 4, Subcodes = 1, 2, 3).

The following DATA subpacket (Code = 4, Subcode = 4) contains information about a "Block Replacement" operation:

```
SUBPACKET DATA = DISP.NEXT_PACKET ;
  FLAGS                :WORD ;
    PRIMARY_RBN        :BIT [0] ;
    SUCCESS             :BIT [1] ;
  LBN                  :LONGWORD ;
  NEW_RBN               :LONGWORD ;
  OLD_RBN               :LONGWORD ;
END_PACKET ;
```

4.6.5.6 Subpackets Declared by DSP5M1/DSP5P1 - This DATA subpacket (Code = 5, Subcode = 1) contains information about a "Memory Parity Error" event:

```
SUBPACKET REGISTER = DISP.NEXT_PACKET NAMED ;
  RESERVED             :WORD ;           ! Mask for Cache Registers
  RESERVED             :WORD ;           ! Mask for Parity CSR's
  P_CSR00              :WORD ;           ! Memory Parity CSR 00
  P_CSR01              :WORD ;           ! Memory Parity CSR 01
  P_CSR02              :WORD ;           ! Memory Parity CSR 02
  P_CSR03              :WORD ;           ! Memory Parity CSR 03
  P_CSR04              :WORD ;           ! Memory Parity CSR 04
  P_CSR05              :WORD ;           ! Memory Parity CSR 05
  P_CSR06              :WORD ;           ! Memory Parity CSR 06
  P_CSR07              :WORD ;           ! Memory Parity CSR 07
  P_CSR08              :WORD ;           ! Memory Parity CSR 08
  P_CSR09              :WORD ;           ! Memory Parity CSR 09
  P_CSR10              :WORD ;           ! Memory Parity CSR 10
  P_CSR11              :WORD ;           ! Memory Parity CSR 11
  P_CSR12              :WORD ;           ! Memory Parity CSR 12
  P_CSR13              :WORD ;           ! Memory Parity CSR 13
  P_CSR14              :WORD ;           ! Memory Parity CSR 14
  P_CSR15              :WORD ;           ! Memory Parity CSR 15
  LOW_ERR              :WORD ;           ! Low Error Address Register
  HIGHERR              :WORD ;           ! High Error Address Register
  CACHERR              :WORD ;           ! Cache Error Register
  CSHCTRL              :WORD ;           ! Cache Control Register
  CSHMAIN              :WORD ;           ! Cache Maintenance Register
  CACHHIT              :WORD ;           ! Cache Hit/Miss Register
END_PACKET ;
```

The following DATA subpacket (Code = 5, Subcode = 2) contains information about an "Unknown Interrupt" event:

```
SUBPACKET DATA = DISP.NEXT_PACKET ;
  VECTOR_OVER_FOUR     :BYTE ;
  LOST_INT              :BYTE ;
END_PACKET ;
```

4.6.5.7 Subpackets Declared by DSP6M1/DSP6P1 - The "Power Recovery" event (Code = 6, Subcode = 1) has no DATA subpacket.

ERROR LOG CONTROL FILE ARCHITECTURE
Notes Module

4.6.5.8 Subpackets Declared by DSP7M1/DSP7P1 - The following DATA subpacket (Code = 7, Subcode = 1) contains information about a "Time Change" event:

```
SUBPACKET DATA = DISP.NEXT PACKET ;  
    NEW TIME           :RSX_TIME ;  
END_PACKET ;
```

The following DATA subpacket (Code = 7, Subcode = 2) contains information about a "System Crash" event:

```
SUBPACKET DATA = DISP.NEXT_PACKET ;  
    CRASH TIME         :RSX TIME ;  
    OP_SYS             :BYTE ;  
    FORMAT_ID          :BYTE ;  
    OP_SYS_ID          :ASCII [4] ;  
    TASK_NAME          :LONGWORD ;  
    TI_DEV             :ASCII [2] ;  
    TI_UNIT            :BYTE ;  
    FLAGS              :BYTE ;  
    KERNEL_APR5        :LONGWORD ;  
    URM                :WORD ;  
        URM_CPU        :FIELD [0:4] ;  
END_PACKET ;
```

The following DATA subpacket (Code = 7, Subcodes = 3, 4) contains information about a "Driver Load" or "Driver Unload" event:

```
SUBPACKET DATA = DISP.NEXT PACKET ;  
    DRIVER_NAME        :ASCII [2] ;  
END_PACKET ;
```

The following DATA subpacket (Code = 7, Subcode = 6) contains information about a "System Message" event:

```
SUBPACKET DATA = DISP.NEXT PACKET ;  
    MESSAGE_LEN        :WORD ;  
    MESSAGE_TEXT       :ASCII [80] ;  
END_PACKET ;
```


CHAPTER 5

CONTROL FILE LANGUAGE GUIDE

This chapter describes the Control File Language used by the Report Generator of the Error Logging System. The preceding chapter describes how to include support for non-DIGITAL devices in the Error Logging System.

5.1 CONTROL FILE OVERVIEW

The control file for the Report Generator (RPT) describes the format of the error log file and the format of reports based on the file. The actions specified are executed for each event to produce a report. The control file specifies the format of the data in an error log packet, and the output format of the report. In addition, the control file specifies information on the accumulation of summary information, how to derive additional information, and the handling of selection criteria for reports.

Control file modules are ASCII text files containing a series of statements written in the Control File Language (CFL). The CFL compiler produces intermediate form modules which are placed in a universal library. This library is the control file.

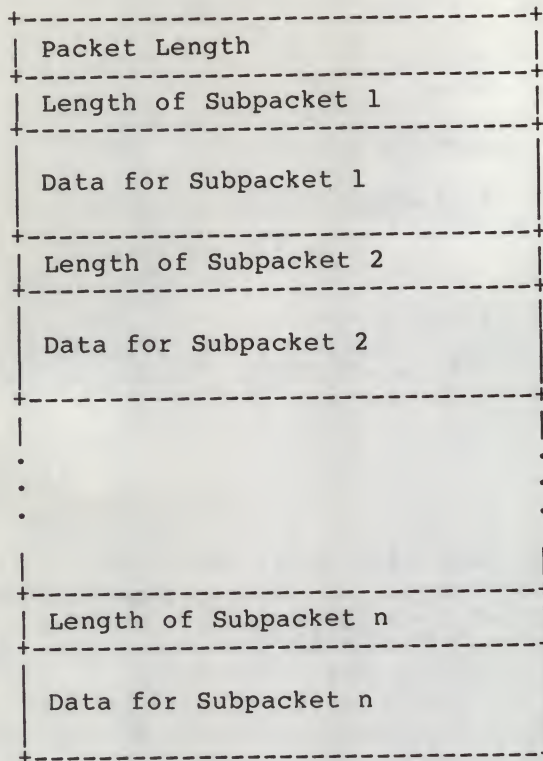
5.1.1 Report Generator General Processing

RPT is an interpreter for the intermediate form modules contained in the control file. RPT processes control file modules which can, in turn, process error log files. The control file module, written in CFL, specifies the processing to be performed on the current packet. The processing usually involves calling subroutines from other modules.

CFL includes primitives that stop packet processing on an error or because the packet does not meet specified selection criteria.

5.1.2 The General Format of an Error Log Packet

Here is the general format of an error log packet:



Each subpacket contains a different type of data. The information in the subpackets, taken together, describes an event logged by the error logging system. The control file contains primitives for describing a subpacket so that its contents can be symbolically manipulated. Other primitives describe the entire packet as a unit.

5.1.3 Control File Language

CFL is a specialized language designed for this application. It is a statement-oriented, block-structured language similar in concept to Pascal and Algol. Unlike BLISS, CFL is not expression-oriented. CFL does differentiate between statements and expressions.

However, CFL has some of the capabilities of BLISS in that expressions can contain conditionals. This feature handles the more complex data formats of error log files. CFL does not include the full set of primitives required of an ordinary general-purpose language. A number of specialized primitives speed up and simplify the handling of common error log data formats.

5.1.4 General Format of Control File Modules

Each control file module contains lines of ASCII text. Each line is a sequence of elements - keywords, variable names, numbers, operators, and the like. Spaces and tabs separate atomic items, such as keywords or names. Excess spaces and tabs are ignored and can be used freely for formatting.

CONTROL FILE LANGUAGE GUIDE

Insert comments in the module text by prefixing them with an exclamation point (!). The compiler ignores any text on a line following an exclamation point.

The three basic elements of CFL are statements, expressions and declarations.

1. A statement describes an action. Statements begin with a keyword and are terminated by a semicolon (;).
2. An expression describes a computation. Expressions are terminated by any nonexpression keyword, or by a comma (,). A nonexpression keyword is a keyword that is not valid in an expression. Expressions can also be terminated like a statement, with a semicolon (;). Expressions can also be included in an expression-list, using conditionals to determine whether or not a given statement or expression is to be executed. Expression-lists consist of a fixed number of expressions, separated by commas (,).
3. A declaration defines the content of packets and subpackets and defines groups of tables for evaluating packets and subpackets. The syntax of a declaration differs for each use. Group declarations start with DECLARE and end with END DECLARE. Table declarations begin with TABLE and end with END TABLE. Dynamic table declarations begin with DYNAMIC TABLE and end with END TABLE. Packet and subpacket declarations begin with PACKET and SUBPACKET, respectively, and end with END_PACKET.

5.1.5 Files

CFL can obtain input from and direct output to any one of a set of files. The files have the following internal names:

- INPUT

The data input file. Data packets are read from the file. The control file can open this file, close it, and read packets from it.

- REPORT

The report output file. Lines of ASCII text are written to this file. The control file can open this file, close it, and write ASCII text to it. Automatic paging is available for this file.

- USER

The user prompting file. The user can be prompted for input, and input read from this file. The control file can open this file, close it, and read ASCII data from this file, with optional prompting.

- COMMAND

The command input file. The user can be prompted for input, and input read from this file. The control file can read ASCII data from this file, with optional prompting.

- ERROR

The error output file. The control file can write ASCII data to this file.

5.2 TYPES AND EXPRESSIONS

RPT types data to allow easy manipulation of error log information. Expressions describe data values used by RPT. This section describes the attributes of supported data types and the format of expressions.

5.2.1 Data Types

CFL supports seven data types: logical, string, numeric, field, pointer, RSX TIME and VMS TIME. The evaluation of an expression results in a value. This value is one of the supported data types. The data type of an expression is determined from its context.

The only automatic conversions are from string values to numeric values, and conversions of numeric values between different numeric types and field types.

The following sections describe the types in detail.

5.2.1.1 LOGICAL Type - The LOGICAL type expresses the Boolean values, TRUE and FALSE. A LOGICAL type is equivalent to a BIT type. No other automatic conversions are performed to or from LOGICAL types. Express the literal values for this type with the keywords TRUE and FALSE.

5.2.1.2 STRING Type - The STRING type represents strings of binary bytes. Literal values for the string type cannot be represented. String operations must be used to construct string literals.

For purposes of conversion, numeric values are considered exactly equivalent to strings. The length of the string is the number of bytes used to represent a value of the numeric type. For example, a WORD is equivalent to a string of length 2. The following equivalences are used:

Type	Equivalent string
BYTE	String of length 1
WORD	String of length 2
LONGWORD	String of length 4
QUADWORD	String of length 8

Strings of length four or less are converted to numeric types by appending leading zero bytes to form a longword. Strings of length five to eight are converted to numeric types by appending leading zero bytes to form a quadword. Strings of length greater than eight are not converted to numeric types.

String declaration requires specification of the maximum size of the string. The syntax for string declaration is:

```
STRING[ size ]
```


If the string is a variable, it can contain any number of elements up to and including the specified maximum. If the string is part of a data declaration, it contains exactly the number of characters specified.

5.2.1.3 ASCII Type - The ASCII type represents character strings. ASCII string literals are represented by character strings enclosed in a pair of apostrophes ('string'). Two successive apostrophes in a quoted string represent a single quoted apostrophe. Therefore the string 'ABC'DE' represents the string literal ABC'DE. The keyword NULL represents the null string. There is no automatic conversion to or from ASCII strings. You cannot use the quotation mark (") to enclose such strings, nor does the quotation mark require flagging.

ASCII string declaration requires specification of the maximum length of the string. Specify the maximum string length as follows:

```
ASCII[ size ]
```

If the ASCII string is a variable, it can contain any number of characters up to the specified maximum. If the ASCII string is a data item, it must contain exactly the number of characters specified.

5.2.1.4 Numeric Types - Numeric data types represent numbers for computation. The numeric types are distinguished only by the length of the bit field used to contain the number. A BYTE is a one-byte field, a WORD a two-byte field, a LONGWORD a four-byte field, and a QUADWORD an eight-byte field. For purposes of conversion, the numeric types are considered equivalent to strings, with the length determined by the type.

The special numeric type VALUE indicates a natural machine value. A VALUE is a WORD on the PDP-11 and a LONGWORD on the VAX-11.

Numeric types have a default output radix of decimal. The syntax for expressing a numeric type is:

```
type option,option,...
```

The valid options are divided into radix options and attribute options. The radix options determine the print radix. They are DECIMAL, OCTAL, HEX, BCD, BINARY, or RAD50.

The attribute options are WIDTH and FILL.

The WIDTH option has the format

```
WIDTH=n
```

and specifies the width of print field in characters.

The FILL option has the format

```
FILL='character'
```

and specifies the fill character.

CONTROL FILE LANGUAGE GUIDE

Here are the default print field width and fill character for each choice of radix:

Radix	Fill	Print Field Width			
		BYTE	WORD	LONGWORD	QUADWORD
DECIMAL	space	3	5	10	20
OCTAL	'0'	3	6	11	22
HEX	'0'	2	4	8	16
BCD	'0'	2	4	8	16
BINARY	'0'	10	20	40	80
RAD50	space	n/a	3	6	12

The default radix is DECIMAL.

For example, here is the specification for a LONGWORD to be printed in BCD using leading spaces and a field six characters wide:

```
LONGWORD BCD,WIDTH=6,FILL=' '
```

The special radix MACHINE is the normal radix used to express values for the host machine. The MACHINE radix is OCTAL for the PDP-11 and HEX for the VAX-11.

You can express numeric literals in a number of ways. A sequence of digits is, by default, interpreted as a VALUE numeric literal. The number is assumed to be decimal. Express a numeric literal of a specified type and radix as follows:

```
# <type indicator> '<character string>'
```

The character string is interpreted according to the specification given by the type indicator. The type indicator is a one- or two-character string specifying the type of the number and the radix in which to interpret the character string. The first character of the type indicator is the type of the number, as shown:

B	Byte
W	Word
L	Longword
Q	Quadword
V	Value

The second character of the type indicator is the radix in which to interpret the character string. If the radix is not specified, the character string is assumed to be decimal. The valid radix indicators are:

A	ASCII
B	Binary
D	Decimal
I	Bit value
O	Octal
R	Radix 50
X	Hexadecimal

A minus sign (-) preceding any character string interpreted as a number indicates the two's complement of that number in the indicated radix, that is, binary, octal, decimal, and hexadecimal.

CONTROL FILE LANGUAGE GUIDE

For example, the character string to represent a byte that contains the octal value "17" would be:

```
#BO'17'
```

while the character string to represent a word containing the value "-16." would be:

```
#W'-16'
```

The bit value radix indicator specifies that the quoted number is a decimal number representing a bit position. The value of the literal is 2 raised to the power of the bit position.

5.2.1.5 Field Types - The field types represent fields of numeric types. The BIT type represents a single bit of a numeric type, and is equivalent to a LOGICAL type. The FIELD type represents a one-or-more-bit field of a numeric type, and is equivalent to a numeric type. A field type is always a field of a numeric variable. There are no literal values for field types.

The syntax for expressing a BIT type is:

```
BIT[ bit number ]
```

The syntax for expressing a FIELD type is:

```
FIELD[ low bit number : field length in bits ]
```

In either case a field type is declared directly following the numeric type of which it is a field.

5.2.1.6 POINTER Type - The POINTER type is a table pointer. Use it to declare variables that temporarily store pointers for later use. The POINTER type cannot be converted to or from any other type. There is no literal representation of the POINTER type.

The value of the POINTER type is specific to a given table. A POINTER variable containing a value specifying a table entry for a given table cannot be used to reference an entry in another table. The variable can, however, be loaded with another value referencing an entry in another table.

5.2.1.7 RSX TIME Type - The RSX_TIME type represents a time in RSX format. RSX time is represented as six sequential bytes, containing the year since 1900, the month, day, hour, minute, and second in that order. This is a compression of the format returned by the Executive GTIM\$ directive.

The RSX_TIME type can only be printed or compared to other RSX_TIME types, or converted using one of the %TIM functions.

5.2.1.8 VMS TIME Type - The VMS_TIME type represents a time in VMS format. VMS time is represented as a quadword containing the time in hundreds of nanoseconds since 17 November 1858.

The VMS_TIME type can only be printed or compared to other VMS_TIME types, or converted using one of the %TIM functions.

5.2.2 Variables

The named variable is the fundamental unit for data manipulation. Named variables are defined in a given module, and available to that module and any modules called by the module. Named variables are declared in named groups. The full name of a variable, that is, the name by which it is referenced, is the name of the group, a period (.), and the name of the variable in the group:

<group name>.<variable name>

The group and variable names cannot be more than 15 characters. Names can include the characters A through Z, the numbers 0 through 9, the dollar sign (\$), and the underscore (_). The leading character of a name must be alphabetic. Use the same syntax to reference data in either packets or subpackets.

The CFL compiler assigns each variable a type through declarations. (See Section 5.4 for a description of the declaration process.) Variables that are not fixed-length, such as ASCII and STRING type variables, are assigned a maximum length as well. The variable can contain any amount of data that fits in its maximum length specified.

A field in the current record of a table can be referenced in the same manner as a variable, as follows:

<table name>.<field name>

The field value referenced is the specified field in the current record of the table. If there is no current record for the table, an error results.

Several special variables provide information about a group, packet, subpacket, or table. Reference the special variables as follows:

<group, [sub]packet or table name>.<special variable name>

The special variables are described below:

- LENGTH

LENGTH is the length of the data in the group in addressable units of the host machine (bytes for the PDP-11 and VAX-11). LENGTH includes all repetitions for repeated data or records for tables.

- POINTER

POINTER returns the current pointer for the specified group. POINTER is not valid for any data structure that would not have a current record context. This includes variables and non-repeated data. There is a current record context for tables and repeated data.

- CONTEXT

CONTEXT returns a logical value. If the specified group has a current record context, the value is TRUE. If the specified group has no current record context, the value is FALSE.

- COUNT

COUNT returns a numeric value representing the number of records in a group. For groups of variables, it is always 1. For packets or subpackets, it is the number of repetitions of the data. For tables it is the number of records in the table.

5.2.3 Literals

Literal values can be assigned symbolic names (See Section 5.6.2 for information on the LITERAL statement). These symbolic names have the same syntax restrictions as variable names. Literal names are considered equivalent to their values in expressions.

5.2.4 Expressions

Expressions describe a computation through a sequence of operands and operators. Operands are variables or literals. Operators direct the computation. Expression evaluation is from left to right, and operator precedence is observed. Use parentheses, (() and ()), to override precedence.

Operators are either unary, which means that they take one operand, or binary, which means that they take two operands. Unary operators can either precede the operand, in which case they are called prefix operators, or succeed the operand, in which case they are called suffix operators. Binary operators are always between the two operands.

Operators are type-specific, that is, they operate between two elements of a specific type to produce a result of a specific type. The elements of an expression can be any of the following:

- Literals

Literals express fixed values of a given type.

- Variables

Variables reference previously computed values.

- Subexpressions in parentheses

Any valid expression enclosed in parentheses can be used as an element to an operator.

- Functions

A function is a predefined computation. (See Section 5.3.)

The following sections describe the operators in detail.

5.2.4.1 String Operators - String operators produce either binary or ASCII string results. The result is a string with the same type as the operand string or strings. The string operators are:

CONTROL FILE LANGUAGE GUIDE

- String concatenation - | binary operator

The string concatenation operator concatenates the first operand with the second. Both strings must be of the same type. For example, the following expression:

```
'ABC' | 'DEF'
```

produces the string:

```
'ABCDEF'
```

- Substring extraction - <n:m> unary suffix operator

The substring extraction operator produces the string formed by character n and the next m elements. String element numbers start with 1. For example, the expression:

```
'ABCDEFGH'<4:3>
```

produces the string:

```
'DEF'
```

Both n and m must be word values.

- Element Extraction - <n> unary suffix operator

The element extraction operator is a special case of the substring extraction operator which extracts the nth element as a single-character substring. For example, the expression:

```
'ABCDEFGH'<4>
```

produces the string:

```
'D'
```

5.2.4.2 Logical Operators - Logical operators perform operations on logical variables, or compare string or numeric variables to yield logical results.

Here are the logical operators:

- Logical AND binary operator

The logical AND operator does a logical AND of the two expressions. For example, the following expression:

```
TRUE AND FALSE
```

produces the logical value FALSE.

- Logical OR binary operator

The logical OR operator does a logical OR of the two expressions. For example, the following expression:

```
TRUE OR FALSE
```

produces the logical value TRUE.

- Logical Exclusive-OR (XOR) binary operator

The logical exclusive-OR operator does a logical exclusive-OR of the two expressions. For example, the following expression:

```
TRUE XOR TRUE
```

produces the logical value FALSE.

- Logical NOT unary prefix operator

The logical NOT operator produces the logical complement of a single variable. For example, the following expression:

```
NOT TRUE
```

produces the logical value FALSE.

There is also a logical operator to produce logical results from numeric expressions:

- Bit extraction - [n] unary suffix operator

The bit extraction operator is TRUE if and only if bit n of the longword expression operand is set. For example, the following expression:

```
#WO'305' [4]
```

produces the logical value FALSE. The binary value of octal 305 is 011000110, with the fourth bit clear. Bits are numbered from 0.

5.2.4.3 Relational Operators - Relational operators compare string, time, or numeric operands. The comparisons are string comparisons if both operands are string or ASCII string operands. The comparisons are numeric comparisons if one operand is numeric and the other is either numeric or string. You cannot compare ASCII string operands and numeric operands. The comparisons are time comparisons if both operands are times of the same type. You cannot compare different types of time.

In numeric comparisons, the larger numeric value is greater.

In string comparisons, CFL stops at the first two characters that don't match and performs an ASCII sort. That is, CFL compares the ASCII values of the characters.

NOTE

Although Z is greater than A in ASCII, an ASCII sort is not the same as an alphabetical sort. Any lowercase letter has greater value than any uppercase letter, for instance, but any alphabetical character has greater value than any numerical character, and so forth. See any standard reference.

CONTROL FILE LANGUAGE GUIDE

If one string is longer than the other and the shorter string has the same leading elements as the longer, the longer string is greater.

In time comparisons, later times are greater.

Here are the relational operators:

- Equality (EQ) binary operator

The equality operator is TRUE if and only if the operands are equal. For example, the following expression:

```
#WD'123' EQ #WD'355'
```

produces the logical value FALSE.

- Inequality (NE) binary operator

The inequality operator is TRUE if and only if the operands are not equal. For example, the following expression:

```
'ABCDEF' NE 'ABC'
```

produces the logical value TRUE.

- Greater-than (GT) binary operator

The greater-than operator is TRUE if and only if the first operand is greater than the second operand. For example, the following expression:

```
123 GT 355
```

produces the logical value FALSE.

- Less-than (LT) binary operator

The less-than operator is TRUE if and only if the first operand is less than the second operand. For example, the following expression:

```
'ABCDEF' LT 'ABCZZZ'
```

produces the logical value TRUE.

- Greater-than-or-equal (GE) binary operator

The greater-than-or-equal operator is TRUE if and only if the first operand is greater than or equal to the second operand. For example, the following expression:

```
45 GE 45
```

produces the logical value TRUE.

- Less-than-or-equal (LE) binary operator

The less-than-or-equal operator is TRUE if and only if the first operand is less than or equal to the second operand. For example, the following expression:

```
'Z' LE 'A'
```

produces the logical value FALSE.

- String-matching (MATCH) binary operator

The MATCH operator compares strings. The strings are examined to determine which is shorter. The shorter string is compared character-by-character to the longer string. If all characters in the shorter string match with characters in the longer string, then the strings are equal and the value is TRUE. This means a null string always matches any other string. For example:

`'ABCDEF' MATCH 'AB'`

produces the logical value TRUE and the expression:

`'ABCDEF' MATCH 'ABCX'`

produces the logical value FALSE.

5.2.4.4 Numeric Operators - Numeric operators operate on numeric variables as unsigned longwords. The numeric operators are:

- Field extraction ([n:m]) suffix unary operator

The field extraction operator produces the longword formed by taking the m-bit field beginning at bit n in the longword. Bit positions are numbered from least significant to most significant, beginning with 0. For example, the expression:

`#WO'357' [3:6]`

produces the octal 35. Octal 357 has the binary value 011101111. Bit three and the next six bits have the binary value 011101, or octal 35.

- Logical SHIFT binary operator

The SHIFT operator produces the first operand shifted by the number of bit positions specified by the second operand. Each left shift of one bit is the equivalent of multiplying by 2 and each right shift of one bit is the equivalent of dividing by 2. Indicate a left shift by making the second operand positive, and a right shift by making the second operand negative. If the second operand is zero, nothing shifts. The shifting is logical shifting; there is no sign extension on right shifts.

For example the following expression:

`#WD'205' SHIFT 2`

produces decimal 820, which is decimal 205 multiplied by 4.

- Multiplication (*) binary operator

The multiplication operator produces the product of the two operands. The result of the multiplication is truncated to the 32 low-order bits.

For example, the expression:

`5 * 3`

produces decimal 15.

- Division (/) binary operator

The division operator produces the integer quotient of the two operands.

For example, the expression:

15 / #B'2'

produces decimal 7.

- Modulus (MOD) binary operator

The MOD operator produces the remainder of the integer division of the two operands.

For example, the expression:

15 MOD 2

produces decimal 1.

NOTE

A division or modulus operation with zero as the divisor causes an error.

- Addition (+) binary operator

The addition operator produces the sum of the two operands. The sum is truncated to the 32 low-order bits.

For example, the expression:

5 + 12

produces decimal 17.

- Subtraction (-) binary operator

The subtraction operator, or minus, produces the difference of the two operands. The difference is truncated to the 32 low-order bits.

For example, the expression:

12 - 3

produces the decimal value 9.

- Negation (-) unary prefix operator

The negation operator, or minus, produces the two's complement of the operand. For example, the expression:

- #B'8'

produces the decimal value -8.

NOTE

The minus is both a unary and binary operator.

CONTROL FILE LANGUAGE GUIDE

Other numeric operators perform bitwise logical operations between two numeric operands. That is, rather than comparing the numeric operands as numbers, these operators compare the numeric operators bit by bit.

Here are the bitwise logical operators:

- Bitwise AND binary operator

The bitwise AND operator produces the bitwise logical AND of the two operands. For example, the following expression:

`#BO'41' AND #BO'3'`

produces the octal value 1. The binary value of octal 41 is 00100001 and the binary value of octal 3 is 00000011. The bitwise AND operation determines that the least significant bit is set in both operands and returns the binary value 00000001, or octal 1.

- Bitwise OR binary operator

The bitwise OR operator produces the bitwise logical OR of the two operands. For example, the following expression:

`#BO'41' OR #BO'3'`

produces the octal value 43. That is, the bitwise OR of the binary values returns the binary value 00100011, or octal 43.

- Bitwise Exclusive-OR (XOR) binary operator

The bitwise XOR operator produces the bitwise logical Exclusive-OR of the two operands. For example, the following expression:

`#BO'41' XOR #BO'3'`

produces the octal value 42. That is, the bitwise exclusive-OR of the binary values returns the binary value 00100010, or octal 42.

- Bitwise complement (NOT) unary operator

The bitwise NOT operator produces the bitwise complement (logical negation) of the operand. For example, the following expression:

`NOT #BO'41'`

produces the octal value 336. That is, the binary value of octal 41 is 00100001 and its complement is 11011110, or octal 3360.

5.2.5 Operator Precedence

Operations occur in the order defined by operator precedence unless overridden using parentheses. Operator precedence in CFL is the same as in most other languages, such as FORTRAN. Operators with higher precedence are evaluated before operators with lower precedence. For example, the expression:

`A + B * C`

CONTROL FILE LANGUAGE GUIDE

is evaluated as $A + (B * C)$ rather than $(A + B) * C$, because the multiplication operator, $*$, has higher precedence than the addition operator, $+$. In general expressions are evaluated from left to right, taking into account operator precedence, unless overridden by parentheses.

Operators are classified into the categories shown in the following table, listed in order of decreasing precedence. The order in which operators are listed within a category is not significant.

Highest precedence class - prefix/suffix unary operators:

-	Numeric negation
NOT	Logical or numeric bitwise negation
[n:m]	Numeric field extraction
[n]	Logical value extraction
<n:m>	Substring extraction
<n>	Element Extraction

Multiplication precedence class - numeric binary operators:

*	Numeric multiplication
/	Numeric division
SHIFT	Numeric logical shifting
MOD	Numeric modulus

Addition precedence class - numeric binary operators:

+	Numeric addition
-	Numeric subtraction

Logical operation class - logical/bitwise logical operators:

AND	Logical and bitwise logical AND
OR	Logical and bitwise logical OR
XOR	Logical and bitwise logical XOR

Relational class - logical comparison operators:

EQ	Equality
NE	Inequality
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to

5.3 FUNCTIONS

Functions provide special computations or special values not otherwise available to a program written in CFL.

A reference to a function in an expression has the format:

% <function >[(argument [= value] , ...)]

Function names have the format % <class name>\$<function name>. Some functions require arguments. Functions return a value of a type that is fixed for a given function. For example, the following function:

%PKT\$IDENT

returns the identification code of the current data packet as an ASCII string.

The following sections list by class name the CFL functions and the values they return.

5.3.1 %CND Functions - Conditional Functions

The %CND functions select one of a set of expressions for evaluation. You can state criteria to select one of the arguments to be evaluated in a given context.

NOTE

All expressions are evaluated before determining a result. This means all expressions must be valid for any possible value of the logical expression. That is, %CND\$IF is not entirely equivalent to an IF-THEN-ELSE statement, and %CND\$SELECT is not entirely equivalent to a SELECT statement.

The functions are:

- %CND\$IF(<logical>,<true exp>,<>false exp>)

Evaluates the specified logical expression. If the expression is true, the true expression is returned as the value of the function. If the expression is false, the false expression is returned as the value of the function.

- %CND\$SELECT(<selector>,<exp else>,<exp 0>,<exp 1>,...)

Evaluates the specified selector expression. If the value of the expression is zero, the exp 0 expression is returned as the value of the function. If the value of the expression is one, the exp 1 expression is returned as the value of the function. In general, if the value of the selector expression is n, the value of exp n is returned as the value of the expression. If no expression is provided corresponding to the value of the selector expression, the value of exp else is returned as the value of the expression.

5.3.2 %CNV Functions - Conversion Functions

The %CNV functions convert expressions to ASCII strings. This is done primarily for printing. The conversions allow specification of the output radix, leading fill character (if any), and number of digits converted.

5.3.2.1 %CNV Functions - Numeric Conversion Functions - The numeric conversion functions convert numeric values to ASCII strings in the radix of the specific function. The syntax of these functions is as follows:

```
%CNV$xxx(<numeric_value>[,<field_width>[,<fill_character>]])
```

where xxx is the radix. If no field_width is specified, the default is 0. If no fill_character is specified, the default is the null character. The field_width and the fill_character control the length of the returned string and justification of the digits in the string.

The numeric conversion functions behave as follows. CFL converts the numeric value to an ASCII string using the appropriate radix, and calculates the number of resulting digits. The following algorithm formats the returned string.

```

if field_width = 0 then
    return a string of length number_of_digits containing only the
        converted digits
else
    if number_of_digits > field_width then
        return a string of length field_width filled with asterisks
    else
        if fill_character = null_character then
            return a string of length field_width with the digits left
                justified and pad the string with trailing blanks
        else
            return a string of length field_width with the digits right
                justified preceded by the specified fill_character

```

- %CNV\$OCTAL(<numeric_value>[,<field_width>[,<fill_character>]])
Converts the number from binary to ASCII octal representation.
- %CNV\$DECIMAL(<numeric_value>[,<field_width>[,<fill_character>]])
Converts the number from binary to ASCII decimal representation.
- %CNV\$DECIMAL_P(<numeric value>[,<field width>[,<fillcharacter>]])

This function is identical to %CNV\$DECIMAL function, except that it appends a decimal point to the end of the output ASCII string. The decimal point is not counted in the field width.

- %CNV\$HEX(<numeric_value>[,<field_width>[,<fill_character>]])
Converts the number from binary to ASCII hexadecimal representation.
- %CNV\$BCD(<numeric_value>[,<field_width>[,<fill_character>]])
This function is identical to %CNV\$HEX. Converts the number from binary to ASCII hexadecimal representation.
- %CNV\$BINARY(<numeric_value>[,<field_width>[,<fill_character>]])
Converts the number from binary to ASCII binary representation.

- `%CNV$MACHINE(<numeric_value>[,<field_width>[,<fill_character>]])`

Converts the number from binary to ASCII representation in the natural machine radix, which is octal for the PDP-11 and hexadecimal for the VAX-11.

- `%CNV$RAD50(<numeric_value>[,<field_width>[,<fill_character>]])`

This function converts a numeric type to an ASCII string using Radix-50 conversion. The numeric value must be a word, longword, or quadword.

5.3.2.2 %CNV Functions - Miscellaneous Conversion Functions - A number of other conversion functions are available:

- `%CNV$STRING(<string>)`

Performs a hexadecimal conversion of the specified string to ASCII.

- `%CNV$RSX_TIME(<RSX time value>[,<fields>])`

Converts the RSX time value to a string of the format: yy-mm-dd hh:mm:ss. The optional fields numeric parameter specifies the number of fields of the date to be converted. To convert only the date, specify 3. To convert the date and time, exclusive of the seconds, specify 5. The default is full date and time expressed in six fields.

- `%CNV$VMS_TIME(<VMS time value>[,<fields>])`

Converts the VMS time value to a string of the format: yy-mm-dd hh:mm:ss. The optional fields numeric parameter specifies the number of fields of the date to be converted. To convert only the date, specify 3. To convert the date and time, exclusive of the seconds, specify 5. The default is full date and time expressed in six fields.

5.3.3 %COD Functions - Encoding Functions

The encoding functions convert ASCII strings into numeric values. Various functions do the conversion using different radices.

- `%COD$OCTAL(<string>)`

Converts the string to a VALUE using octal radix. The string may contain only the digits 0 through 7 and optional leading spaces or the minus (-).

- `%COD$DECIMAL(<string>)`

Converts the string to a VALUE using decimal radix. The string may contain only the digits 0 through 9 and optional leading spaces or the minus (-).

- `%COD$HEX(<string>)`

Converts the string to a VALUE using hexadecimal radix. The string may contain only the digits 0 through 9 and A through E, and optional leading spaces or the minus (-).

- **%COD\$BCD(<string>)**

Same as %COD\$HEX. Converts the string to a VALUE using hexadecimal radix. The string may contain only the digits 0 through 9 and A through E, and optional leading spaces or the minus (-).

- **%COD\$BINARY(<string>)**

Converts the string to a VALUE using binary radix. The string may contain only the digits 0 and 1, and optional leading spaces or the minus (-).

- **%COD\$MACHINE(<string>)**

Converts the string to a VALUE using the natural radix for the MACHINE, which is OCTAL for the PDP-11 and HEX for the VAX-11.

- **%COD\$RSX_TIME(<string>)**

Converts the string to a date in RSX format. The string must be of the form dd-mmm-yy [hh:mm[:ss]]. The date and time can occur in either order; the seconds (:ss) are optional. The default for the time fields is 00:00:00.

- **%COD\$VMS_TIME(<string>)**

Converts the string to a date in VMS format. The string must be of the form dd-mmm-yy hh:mm:ss. The date and time can occur in either order; the seconds (:ss) are optional.

5.3.4 %COM Functions -Computational Functions

- **%COM\$AND(<numeric expression> , <numeric expression>)**

Returns the logical AND of the two numeric expressions. Both expressions must be machine VALUES or shorter. This function is used primarily for overlay reasons on the PDP-11.

- **%COM\$HARDWARE(<numeric expression>)**

Returns the ASCII character corresponding to the numeric expression in the DEC hardware alphabet, which is ABCDEFHJKLMNPRSTUV, numbered from 0 through 17. A %COM\$HARDWARE(0) returns an "A".

- **%COM\$LONGWORD(<value> , <bit> , <value> , <bit> ...)**

Returns a LONGWORD value. Each value is shifted by the specified number of bits and then all the values are ORed.

- **%COM\$NEGATE(<value>)**

Returns the negative of the specified value. This is the two's complement of the value.

- **%COM\$NULL(<expression>)**

Returns a TRUE if the result of the expression is a value of length zero, a FALSE if the result of the expression is a value with length other than zero.

5.3.5 %CTL Functions - RPT Control

- **%CTL\$OPEN(<file>,<file spec>,<default spec>)**
 Opens the file using the file specification and the default file specification. The value of the function is the fully qualified file specification for the file.
- **%CTL\$STATUS(<file>)**
 Returns the value TRUE if the file is open and FALSE if the file is not open.
- **%CTL\$FILE_STATUS**
 Returns the numeric status value returned by the file system after the last file open operation.
- **%CTL\$EOF(<file>)**
 Returns the value TRUE if the specified file is at EOF, FALSE if the file is not at EOF.
- **%CTL\$CLOSE(<file>)**
 Closes the file. The value of the function is the number of records written to the file.
- **%CTL\$INPUT(<low> , <high>)**
 Sets the lowest and highest packets to be processed by RPT. Returns a TRUE if both packet specifications are syntactically correct and a FALSE if either is not syntactically correct.

 This implicitly sets the processing direction, as well, because if the high packet is lower than the low packet, the file is processed backwards. A null packet specification takes the default, the beginning of file for the low packet and the end of file for the high packet.

5.3.6 %LOK Functions - Lookahead Functions

The %LOK functions obtain information in undeclared data packets or subpackets. There is a %LOK function for each of the data types supported for lookahead. All offsets are byte offsets. Here are the functions:

- **%LOK\$BYTE(<subpacket_number>,<offset>)**
 Returns the specified byte from the current data packet. The subpacket number is the number of the subpacket from which the data is to be obtained. If the subpacket number is zero, the data is obtained from the packet itself. The offset is the byte offset in the subpacket for the data item.
- **%LOK\$WORD(<subpacket_number>,<offset>)**
 Returns the specified word from the current data packet. The subpacket number is the number of the subpacket from which the data is to be obtained. If the subpacket number is zero, the data is obtained from the packet itself. The offset is the byte offset in the subpacket for the data item.

CONTROL FILE LANGUAGE GUIDE

- `%LOK$LONGWORD(<subpacket_number>,<offset>)`

Returns the specified longword from the current data packet. The subpacket number is the number of the subpacket from which the data is to be obtained. If the subpacket number is zero, the data is obtained from the packet itself. The offset is the byte offset in the subpacket for the data item.

- `%LOK$LENGTH(<subpacket_number>)`

Returns the length of the data in the specified subpacket. The subpacket number is the number of the subpacket whose length is to be returned. If the subpacket number is zero, the length of the data packet is returned.

5.3.7 %PKT Functions - Packet Information

The %PKT functions obtain information about the current packet:

- `%PKT$MODULE(<module name>)`

Returns the value TRUE if the specified module exists in the control file, and FALSE if it does not exist.

- `%PKT$IDENT`

Attempts to get the next packet from the input file in the range specified by `%CTL$INPUT` and makes it the current packet. If no more packets exist within that range, a null string is returned. Otherwise, `%PKT$IDENT` returns the current packet identification as a fixed-length ASCII string of eight characters.

5.3.8 %RPT Functions - Report Control

The %RPT functions control report generation:

- `%RPT$PAGE_SIZE(<lines>)`

The default page size is 57 lines of text plus headers and a form feed. `%RPT$PAGE_SIZE` changes the number of lines per page to the specified value. If the value is zero, the page size is infinite. The function returns the previous number of lines per page before the function was executed.

- `%RPT$PAGE_DEFAULT`

Returns the default number of lines per page of RPT, which is decimal 57.

- `%RPT$PAGE_CURRENT`

Returns the current number of lines per page.

- `%RPT$PAGE_REMAINING`

Returns the number of lines remaining on the current page.

- **%RPT\$LINE_SKIP(<interval>,<lines>)**

Causes RPT to skip the specified number of lines every interval number of lines. If the interval is zero, automatic line skipping is suppressed. The function value is the previous interval.

- **%RPT\$LINE_REMAINING**

Returns the number of lines remaining in the current interval.

- **%RPT\$COMMAND**

Returns the command line as a string.

- **%RPT\$IDENT**

Returns RPT ident as a string.

- **%RPT\$STATUS(<status>)**

Sets the exit status of RPT to the specified status value if it is more severe than the current exit status. If it is not more severe, no action is taken. The actual status value is determined by the control files using this function based on the value given by the <status> argument. A status value is considered a SUCCESS, or TRUE, status if the low bit is 1, and a FAILURE, or FALSE, status if the low bit is 0.

The following algorithm, where NEW_STATUS is the value of the <status> argument and EXIT_STATUS is the current exit status, is used to update the exit status:

```

IF NEW_STATUS
THEN
  BEGIN
    IF EXIT_STATUS AND ( NEW_STATUS GT EXIT_STATUS )
    THEN
      SET EXIT_STATUS TO NEW_STATUS ;
    END ;
  ELSE
    BEGIN
      IF EXIT_STATUS OR ( NEW_STATUS GT EXIT_STATUS )
      THEN
        SET EXIT_STATUS TO NEW_STATUS
      END ;
    
```

The function returns to original value of EXIT_STATUS rather than the potentially updated EXIT_STATUS.

5.3.9 %STR Functions - String Handling

The %STR functions manipulate ASCII and binary strings:

- **%STR\$TRAIL(<string>,<element>)**

Removes all trailing repetitions of the specified element from the specified string. The value of the function is the original string without the trailing characters.

- **%STR\$LENGTH(<string>)**

Returns the length of the specified string as a numeric.

CONTROL FILE LANGUAGE GUIDE

- `%STR$PARSE(<string>,<pointer>,<control>)`

`%STR$PARSE` performs a simple parse by returning a pointer to the end of the substring beginning at the specified pointer position in the string and terminated by any of the characters in the control string or by the end of the string.

- `%STR$QUOTE(<string>,<pointer>,<control>,<quote>)`

`STR$QUOTE` performs a simple parse with quote characters. `STR$QUOTE` works the same as `STR$PARSE` except `STR$QUOTE` handles quote characters. The quote argument is a character string of two characters. For clarity, the two characters should match in some way, but this is not required. If the first character of the pair is encountered, checking for control characters stops until the second character of the pair appears. For example, the quote string '<>' causes anything between a left-angle bracket (<) and a right-angle bracket (>) to be considered as "quoted" and treated as a unit.

- `%STR$REMAINING(<string>,<pointer>)`

Returns the substring of the specified string consisting of all characters including and following the specified pointer position.

- `%STR$MATCH(<string>,<string>)`

Performs an element-by-element comparison of the two strings. The comparison continues only as long as there are elements to compare. That is, with strings of different lengths, the comparison stops with the last element in the shorter string. `%STR$MATCH` returns TRUE if the elements match and FALSE if they do not.

- `%STR$SEARCH(<string>,<pointer>,<string>)`

Searches the first string, beginning at the specified pointer position, for the second string. The pointer returns to the position in the first string at which the second string begins. If the second string isn't found, `%STR$SEARCH` returns a zero.

- `%STR$PAD(<string>,<paddingstring>,<lead>,<trail>)`

`%STR$PAD` creates a new string consisting of the specified string padded with the single-character padding string. The lead and trail numeric expressions specify how many padding characters you wish to lead or trail the original string.

- `%STR$FILE(<string>,<pointer>)`

`%STR$FILE` assumes the pointer is at the beginning of file specification. It returns a pointer to the character following the file specification. If the string pointed to is not a valid file specification, `%STR$FILE` returns a zero.

- `%STR$UPCASE( <value> )`

Returns the specified string with all lowercase letters converted to uppercase.

- `%STR$CHAR( <value> )`

Returns the character corresponding to the specified value.

5.3.10 %TIM Functions - Time Handling

The %TIM functions manipulate times. Remember that the time values include both date and time unless otherwise noted. Here are the %TIM functions:

RSX Time Functions:

- %TIM\$RSX_CURRENT
Returns the current date and time as a value in RSX format.
- %TIM\$RSX_DATE(<RSX time value>)
Returns the date only in RSX format.
- %TIM\$RSX_VMS(<VMS time>)
Returns an RSX time value corresponding to the specified VMS time.
- %TIM\$RSX_NULL
Returns a null RSX time value. This value prints as all blank spaces.

VMS Time Functions:

- %TIM\$VMS_CURRENT
Returns the current date and time as a value in VMS format.
- %TIM\$VMS_DATE(<VMS time value>)
Returns the date only in VMS format.
- %TIM\$VMS_PLUS(<VMS time>,<days>)
Returns a VMS time value containing the specified time plus the specified number of days.
- %TIM\$VMS_MINUS(<VMS time>,<days>)
Returns a VMS time value containing the specified time minus the specified number of days.
- %TIM\$VMS_RSX(<RSX time>)
Returns a VMS time value corresponding to the specified RSX time.
- %TIM\$VMS_NULL
Returns a null VMS time value. This value prints as all blank spaces.

5.3.11 %USR Function - User I/O Function

The %USR function performs input and output to and from the user of RPT:

- `%USR$STRING(<file>,<prompt>,<maximum length>)`

Writes out the prompt string if the specified file is a terminal, and reads a string input whose maximum length is specified by the length parameter. If the specified output file is not a terminal, there is no prompt and only the read is performed.

5.4 DECLARATIONS

This section describes the declaration of variables and data items. A declaration includes print-formatting information along with the definition of data items; it is different from declaration in most languages.

5.4.1 Scope of Declarations

Data items can be referenced during the scope of the declaration, that is, from the point they are declared until the declaration is discarded. If a declaration is made in a given procedure, data items can be referenced in the defining procedure or any procedure called by it. What about coroutine interactions.

5.4.2 DECLARE Statement

The DECLARE statement begins the declaration of a block of variables. Here is the format:

```
DECLARE <group name> [ NAMED ] ;
    <variable name> : <type> [: <print expressions>] ;
    <variable name> : <type> [: <print expressions>] ;
    <variable name> : <type> [: <print expressions>] ;
END_DECLARE ;
```

The group name is the prefix name by which the group variables are referenced. Each of the group variables is referenced using the name:

`<group name>.<variable name>`

The variable type is one of the RPT data types: LOGICAL, STRING, ASCII, NUMERIC, FIELD, RSX_TIME or VMS_TIME.

The optional NAMED qualifier specifies that the symbol names are to be kept and used with the WRITE_GROUP statement FORMAT clause qualifiers that print a symbol name.

The optional print expressions specify expressions to be evaluated and printed if the group is printed using the WRITE_GROUP statement. If you specify more than one expression, separate them by commas.

The print information consists of one or more expressions separated by commas. If the WRITE_GROUP statement is used to print the data group, the print expressions are evaluated and printed.

Declaration of a numeric type can be followed by declaration of one or more field types. The field types are considered fields of the preceding numeric type declaration.

CONTROL FILE LANGUAGE GUIDE

Print information for field types is handled specially. The variable names for field types do not appear when the `WRITE_GROUP` statement is used, and the print expressions following a field type declaration are considered to apply to the preceding numeric type. Of the print expressions following a field type, one is selected based on the field value.

For `BIT` fields, the first print expression is used if the `BIT` is `TRUE`, and the second if the `BIT` is `FALSE`.

For `FIELD` fields, the first print expression is used if no other print expression applies. The second print expression is used if the `FIELD` value is zero, the third if the field value is one, the fourth if the field value is two, and so on.

When a print expression is printed for a field type, it is printed in the following format:

```
p[h:l] ttt...ttt
```

using the following symbols:

- p - The leading character of the print expression. This appears as a prefix to the print field.
- h - The high bit number of the `FIELD`, or the bit number of the `BIT`. The brackets are printed.
- l - The low bit number of the `FIELD`. The "l" field and the leading colon are printed as blanks for `BITs`.
- t - The trailing characters of the print expression. The trailing characters are any characters following the first character.

For example, take the following definition:

```
DECLARE EXAMPLE :  
  VARIABLE_1 : WORD :  
  : FIELD [6:2] : '*The value of this field is 2 or  
3',  
  ' The value of this field is 0' ,  
  ' The value of this field is 1' ;
```

When `EXAMPLE` is printed with a `WRITE_GROUP` statement, the field will be printed as follows, depending on whether bits 6 through 7 contain the value 0, 1, 2, or 3:

```
[ 7: 6]      The value of this field is 0  
[ 7: 6]      The value of this field is 1  
*[ 7: 6]      The value of this field is 2 or 3
```

NOTE

The field is declared in the form `[6:2]`, meaning that it starts at bit 6 and is two bits long. However, the print format is expressed in the form `[ 7: 6]`, meaning that it consists of bits 6 through 7.

5.4.3 PACKET Statement

The PACKET statement declares an input data packet. Here is the format:

```
PACKET <name> [ REPEATED ] [ NAMED ] ;
      <name> : <type> : <print information> ;
      <name> : <type> : <print information> ;
      <name> : <type> : <print information> ;
END_PACKET ;
```

The REPEATED data attribute is optional. A PACKET without this attribute specifies a single packet. The REPEATED specifies that the data in the packet is repeated. The number of repetitions is computed by dividing the packet length by the length of the data items. Note that the items must be referenced as for a DYNAMIC_TABLE; they cannot be referenced directly.

The optional NAMED qualifier specifies that the symbol names are to be kept and used with the WRITE_GROUP statement FORMAT clause qualifiers that print a symbol name.

Declaration of data defines the special variable LENGTH, referenced as the data items themselves would be referenced:

```
<data group name>.LENGTH
```

Note that the length can be referenced directly, even for a REPEATED data group.

Each of the data item names is declared, along with the type and print information. The name is the name by which the data element is referenced.

Here is the format for a data element reference:

```
<data group name>.<data element name>
```

The data item type is declared as specified in Section 5.2.1, Data Types.

The special variable name RESERVED in place of an element declaration specifies a sequence of undefined values. The type declaration specifies the length of the undefined area. This cannot be a field type. The syntax is as follows:

```
PACKET <name> <data organization> ;
      <name> : <type> : <print information> ;
      RESERVED : <type> ;
      <name> : <type> : <print information> ;
END_PACKET ;
```

A RESERVED declaration in a PACKET or SUBPACKET indicates to the compiler (and RPT) that the area is currently unused, but to use its length in determining the size of the PACKET or SUBPACKET and the offsets of elements following the RESERVED declaration. Use RESERVED either to reserve space for future use or to force word-boundary alignments.

Note that an element name could be used in these situations, but that RESERVED serves as a documentation aid, and saves having to define unique element names if there are multiple unused areas in a packet or subpacket.

5.4.4 SUBPACKET Statement

The SUBPACKET statement declares an input data SUBPACKET. Here is the format:

```
SUBPACKET <name> = <expression> <data attribute> [ NAMED ] ;
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
END_PACKET ;
```

The attribute, if present, is REPEATED. A SUBPACKET without an attribute specifies a single subpacket. The leading REPEATED attribute specifies that the data in the subpacket is repeated. The number of repetitions is computed by dividing the subpacket length by the length of the data items. Note that the items must be referenced as for a DYNAMIC_TABLE; they cannot be referenced directly.

The optional NAMED qualifier specifies that the symbol names are kept and used with the WRITE_GROUP statement FORMAT clause qualifiers that print a symbol name.

The handling of SUBPACKET is otherwise the same as for PACKET.

5.4.5 Conditional Declarations

RPT provides a mechanism for conditional declaration of data items. Conditional declaration can only be used for data, that is, PACKET and SUBPACKET declarations. FIELD and BIT declarations cannot cross conditionals. All FIELD and BIT declarations must be in the same conditional as their data item.

The conditionals allowed in declarations are as follows:

- IF

IF has the following syntax:

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;

IF <expression>
THEN
      BEGIN
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
      END
ELSE
      BEGIN
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
      <name> : <type> [: <print information>] ;
      END
END_IF ;

<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
```

If the expression is TRUE, the THEN clause is defined. If the expression is FALSE, the ELSE clause is defined. The ELSE clause is optional.

- CASE

CASE has the following syntax:

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
```

CASE <expression> OF

```
[ <expression> , <expression> , ... ] :
```

```
BEGIN
```

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
END
```

```
[ <expression> , <expression> , ... ] :
```

```
BEGIN
```

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
END
```

```
[<expression> , <expression> , ... ] :
```

```
BEGIN
```

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
END
```

```
ELSE
```

```
BEGIN
```

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
END
```

```
END_CASE ;
```

```
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
<name> : <type> [: <print information>] ;
```

The expression in the CASE statement is evaluated and the expression lists searched to find a matching expression value. If a match is found the declaration is made. If no matching expression value is found, the optional ELSE clause is executed. Otherwise, an error occurs.

5.5 ACTION STATEMENTS

Action statements perform processing. CFL has a limited set of action statements because it is a simple, special-purpose language. The statements provided have capabilities designed to make the handling of error log data as simple as possible.

5.5.1 SET Statement

SET sets the value of a variable to the results of a computation. Here is the format:

```
SET <variable name> TO <expression> ;
```

The expression is evaluated using the type of the specified variable, and the variable is set to the value of the expression.

5.5.2 INCREMENT and DECREMENT Statements

INCREMENT and DECREMENT adjust the value of numeric variables of length VALUE or less by a value. The value defaults to 1. The format of the statements are:

```
INCREMENT <variable name> [ BY <numeric expression> ] ;
```

```
DECREMENT <variable name> [ BY <numeric expression> ] ;
```

The value of the variable is increased by the value of the numeric expression for the INCREMENT statement, and decreased by the value of the numeric expression for the DECREMENT statement.

5.5.3 WRITE Statement

WRITE writes information to a specified output. Here is the format:

```
WRITE <expression> , ... TO <output> FORMAT <format> ;
```

The expressions are printed in the order specified. The optional output clause can contain either the specifications REPORT or ERROR. ERROR directs output to the invoking terminal. REPORT is the default and directs output to the output report. The REPORT file must be open. The ERROR file is always open. The format is described in section 5.10, Print Formatting.

5.5.4 WRITE_GROUP Statement

WRITE_GROUP writes a decoded block of data. The data definitions define the formatting. Here is the format:

```
WRITE_GROUP <group name> TO <output> FORMAT <format> , <format> ;
```

The group name is the name of the group of variables or data items to be written. The optional output specification is the same as for the WRITE statement, and has the same defaults. The format is described in Section 5.10, Print Formatting.

The group name can be followed by a symbol list:

```
<group name> <symbol name>,<symbol name>,...
```

In this case only the specified symbols are listed.

The first FORMAT clause is for printing all data items, the second FORMAT clause is for printing all BIT and FIELD items.

5.5.5 DECODE Statement

The DECODE statement performs specialized declaration-to-text translation. The statement has the form:

```
DECODE <variable name> = <group name> ;
      <data item> [ <bit number> ] ;
      <data item> [ <bit number> ] ;
      <data item> [ <bit number> ] ;
END_DECODE ;
```

Each of the data items must be a data item in the specified group. The bit numbers are numbers of bits in the data item. The DECODE statement processes the data items in the order specified, checking the specified bits. If a bit is found to be TRUE, the corresponding bit-to-text translation for that bit is performed, and the result returned in the specified variable. This completes the statement. A data item can be preceded by a NOT, which indicates the specified bit must be FALSE for the bit-to-text translation to be performed. If no bit-to-text translation is found, the null string is returned.

5.6 CONTROL STATEMENTS

Control statements direct RPT. Control statements define and invoke procedures and control termination of the procedure. Other control statements also conditionally control the execution of statements in a procedure. These are called conditional statements.

5.6.1 MODULE Statement

MODULE declares the name of the module being compiled. Here is the format:

```
MODULE <name> <ident> ( attribute , attribute , ... ) ;
```

MODULE must be the first statement in any module. Each module must end with an END_MODULE ; statement.

The module name specifies the name to be used when the module is inserted into the control file library.

The module ident is a quoted string to be inserted in the module header.

The optional module attributes specify information to be used in processing the module. Two attributes are recognized:

- KEEP

KEEP specifies that if a module cache is used by RPT, the module should be kept because it is likely to be used again.

- FLUSH

FLUSH specifies that if a module cache is used by RPT, the module should be flushed because it is unlikely to be used again.

5.6.2 LITERAL Statement

LITERAL assigns a name to a literal value. All LITERAL statements in a module must precede any PROCEDURE statement. The format of a LITERAL statement is:

```
LITERAL <group>.<name> = <compiletime constant expression> ;
```

The name is equivalent to the value of the compiletime constant expression, and can be used in any expression or compiletime constant expression to represent the specified value.

5.6.3 CALL Statement

CALL invokes a subroutine. CALL has the following format:

```
CALL [ MODULE <module name expression> ]  
      PROCEDURE <procedure name expression>  
      [ COROUTINE <procedure name expression> ] ;
```

The optional module name expression specifies the module to be called. If the module name is not specified, the specified procedure is assumed to be in the current module. The procedure name specifies the procedure to call.

The optional COROUTINE argument specifies that the two called procedures are coroutines. The first called procedure is specified as in the normal form of a CALL statement. The procedure specified using the COROUTINE keyword is executed first. That procedure can then execute a statement that passes control to the other procedure specified in the CALL. None of the declarations are lost.

The two procedures can trade control back and forth using the COROUTINE statement. Each time a COROUTINE statement is executed, the other procedure resumes execution from the point of the last COROUTINE statement. If one of the procedures returns, control passes to the other as if a COROUTINE statement were executed. When both have completed, the coroutines exit to the caller.

5.6.4 RETURN Statement

RETURN forces a return from a procedure to the calling procedure. RETURN is optional at the end of a procedure. The format of the RETURN statement is:

```
RETURN ;
```

This terminates the current procedure and returns control to the calling procedure.

5.6.5 PROCEDURE Statement

PROCEDURE declares the beginning of a procedure. It has the following format:

```
PROCEDURE <name> <statement block> ;
```

The procedure name cannot be more than 15 characters. Names can include the characters A through Z, the numbers 0 through 9, the dollar sign (\$), and the underscore (_). The leading character of a name must be alphabetic.

The statement block is executed as the named procedure.

5.6.6 IF-THEN-ELSE Statement

IF-THEN-ELSE is the most basic conditional statement. Other conditional statements are provided to simplify the handling of common situations that would be cumbersome with IF-THEN-ELSE. The IF-THEN-ELSE has the following format:

```
IF <logical expression> THEN <block> ;
    [ELSE <block>;] END_IF ;
```

If the logical expression is TRUE, the block following the THEN statement is executed. If the logical expression is FALSE, the block following the ELSE statement is executed. The ELSE clause is optional. If it is not specified, no action is performed if the expression is FALSE.

Each block consists of a single statement. Using BEGIN-END, a block can contain a compound statement. See Section 5.6.11 on BEGIN-END statements for a description of how to use BEGIN-END statements to make multiple statements appear as a single logical entity to conditional statements.

5.6.7 CASE Statement

CASE selects one of a set of possible outcomes based on an expression. The format of a CASE statement is as follows:

```
CASE <expression> OF
    [ <expression> , <expression> , ... ] : <block> ;
    [ <expression> , <expression> , ... ] : <block> ;
    [ <expression> , <expression> , ... ] : <block> ;
[ELSE <block> ;]
END_CASE ;
```

This executes the block corresponding to the first expression equal to the selector numeric expression. If ELSE is specified, it is executed if no expression matches.

5.6.8 SELECT Statement

SELECT is a special case of CASE. SELECT selects one of a given set of blocks. The general format is:

```
SELECT <numeric expression> OF
    <block> ;
    <block> ;
    <block> ;
[ELSE <block> ;]
END_SELECT;
```


SELECT selects the nth block, where n is the value of the numeric expression and is greater than or equal to 1. If the last block is preceded by ELSE, the block is executed if and only if the value of the numeric expression exceeds the number of blocks supplied.

5.6.9 WHILE/UNTIL/DO Statements

The WHILE/UNTIL/DO statements control conditional looping. To specify a conditional loop, specify a block of statements to be conditionally executed and an expression to control the execution.

The DO statement specifies the block of statements to be conditionally executed.

The WHILE statement specifies an expression to be considered satisfied if it is TRUE.

The UNTIL statement specifies an expression to be considered satisfied if it is FALSE.

A DO statement must be specified with a WHILE or UNTIL statement. The block of statements specified by the DO statement is executed until the condition specified by the WHILE or UNTIL statement is no longer satisfied.

If DO WHILE or DO UNTIL is specified, the DO is executed once before testing the condition. If WHILE DO or UNTIL DO is specified, the condition is tested first before executing the DO statement. This leads to the following statement combinations:

```
DO <block> WHILE <expression> ;
```

Executes the block once, then evaluates the expression. If the expression is TRUE, the block is repeated. If the expression is FALSE, execution continues following the WHILE statement.

```
DO <block> UNTIL <expression> ;
```

Executes the block once, then evaluates the expression. If the expression is FALSE, the block is repeated. If the expression is TRUE, execution continues following the UNTIL statement.

```
WHILE <expression> DO <block>
```

Evaluates the expression. If the expression is TRUE, the block is executed and the process repeated. If the expression is FALSE, execution continues following the DO statement.

```
UNTIL <expression> DO <block>
```

Evaluates the expression. If the expression is FALSE, the block is executed and the process repeated. If the expression is TRUE, execution continues following the DO statement.

5.6.10 LEAVE Statement

LEAVE immediately terminates the current DO statement. The control expression in the associated UNTIL or WHILE statement is considered satisfied and is not reevaluated.

5.6.11 BEGIN-END Statement

BEGIN-END statements force a compound statement to be treated as one statement for purposes of conditionals. For example, to process two statements in the THEN clause of an IF statement, use the the following construct:

```

IF <logical expression>
THEN
    BEGIN
        <statement 1> ;
        <statement 2> ;
    END ;
ELSE
    <statement> ;
END_IF ;

```

5.6.12 Lexical Conditionals

Lexical conditionals perform conditional handling at compilation. A lexical conditional is valid wherever a statement is valid. Lexical conditionals have the following format:

```

$IF <compiletime constant expression>
$THEN
    <statement block>
[$ELSE
    <statement block>]
$END_IF

```

The \$ELSE block is optional. If the compiletime constant expression is TRUE, everything in the \$THEN block is compiled and the \$ELSE block is not compiled. If the compiletime constant expression is FALSE, the \$THEN block is not compiled and the \$ELSE block, if present, is compiled.

Lexical conditionals can be nested to any level.

5.7 TABLES

The table is one of the fundamental units of data organization for RPT. RPT uses tables to structure large amounts of data to be referenced during report generation.

5.7.1 Table Structure

Tables are sets of similar records containing fields by which the records can be referenced. Tables, and the data in them, can either be declared statically as part of the definition of a given control file module, or dynamically during the operation of RPT. Static tables hold reference data, while dynamic tables store information computed during the operation of RPT.

Each record in a table is a sequence of named fields. The definition of the table defines the names of the fields and their sequence.

Reference tables by name. Table names follow the ordinary rules for naming groups. The name cannot be more than 15 characters. Names can

include the characters A through Z, the numbers 0 through 9, the dollar sign (\$), and the underscore (_). The leading character of a name must be alphabetic.

Fields in a table are also named. Field names follow the same rules.

Table entries are manipulated by setting the current entry pointer for a table, and then using either the table manipulation statements or simple variable references to read or modify the data in the table.

The following sections describe each of the table-definition and table-manipulation statements in detail.

5.7.2 TABLE Statement

TABLE defines a static table. The format of the statement is:

```
TABLE <table name> ;
    <name> : <type> [: <print expressions>] ;
    <name> : <type> [: <print expressions>] ;
    <name> : <type> [: <print expressions>] ;
BEGIN_TABLE
    <value> , <value> , <value> , ... ;
    <value> , <value> , <value> , ... ;
    <value> , <value> , <value> , ... ;
END_TABLE ;
```

The declaration list following the TABLE statement specifies each of the fields, their types, and print information. The format is the same as for DECLARE. The list values are individual compile-time constant expressions separated by commas. Each sequence of list values separated by commas (,) and terminated by a semicolon (;) represents one TABLE record. Each table value must be of the same type as the corresponding declaration from the declaration list. TABLE records cannot be modified at run time.

5.7.3 DYNAMIC_TABLE Statement

DYNAMIC_TABLE declares a dynamic table. The format of a DYNAMIC_TABLE statement is as follows:

```
DYNAMIC_TABLE <table name> ;
    <name> : <type> [: <print expressions>] ;
    <name> : <type> [: <print expressions>] ;
    <name> : <type> [: <print expressions>] ;
END_TABLE ;
```

Records are placed in the DYNAMIC_TABLE at run time through use of the PUT statement, and can be modified by some of the POINTER statements.

5.7.4 FILE Statement

FILE is identical to DYNAMIC_TABLE. It is included for compatibility only and you should use DYNAMIC_TABLE. The FILE statement may be removed in a future release.

CONTROL FILE LANGUAGE GUIDE

FILE declares a dynamic table. The format of a FILE statement is as follows:

```
FILE <table name> ;  
    <name> : <type> [: <print expressions>] ;  
    <name> : <type> [: <print expressions>] ;  
    <name> : <type> [: <print expressions>] ;  
ENDFILE
```

Records are placed in the FILE dynamically at run time through use of the PUT statement and can be modified by some of the POINTER statements.

5.7.5 POINTER Statement

POINTER adjusts the current pointer for a table using the following syntax:

```
POINTER <table name> <action> <optional pointer variable> ;
```

Each of the actions is described below:

- FIRST

Sets the current table pointer to the first record of the table. If there is no next record, then the current table pointer is set to null (see RESET).

- NEXT

Sets the current table pointer to the next record of the table. If there is no next record, then the current table pointer is set to null (see RESET).

- PREVIOUS

Sets the current table pointer to the previous record of the table. If you back up past the beginning, then the table pointer is set to null (see RESET).

- RESET

Sets the current table pointer to null, that is, there is no table pointer.

- LOAD <pointer variable>

Sets the current table pointer to the value of the pointer variable.

- CLEAR

The specified table must be a DYNAMIC_TABLE. Deletes all records from the DYNAMIC_TABLE and sets the current table pointer to null (see RESET).

- DELETE

The specified table must be a DYNAMIC_TABLE. Deletes the current record and advances the pointer to the next record. If there is no next record, then the current table pointer is set to null (see RESET).

- **MOVE <pointer variable>**

The specified table must be a FILE. The record pointed to by the pointer variable is moved to the current record position and the current record and all following records are moved up one record. This is used mainly for sorting records in a FILE.

5.7.6 FIND Statement

FIND finds a record in a table using one or more key values. The format of the FIND statement is:

```
FIND <table> <field>=<value> , ... SELECT <expression> ;
```

The table is searched until an entry with all specified fields having the specified value is encountered. Tables are searched sequentially from the current pointer position. If no record is found, the current pointer for the table is set to null.

If you specify the optional SELECT clause, a record does not satisfy the search criteria unless the select expression, evaluated with the current record for the table set to the specified record, is TRUE.

5.7.7 PUT Statement

PUT creates a new record in a table. The specified table must be a DYNAMIC_TABLE. PUT has the following format:

```
PUT <table> <field>=<expression> , ... ;
```

Sets the specified fields of the record to the values of the specified expressions. Note that all fields must be specified; none of the fields of the record are optional.

5.8 LISTS

This section describes the expression-list-handling facilities of CFL.

5.8.1 LIST Statement

LIST declares a list of expression groups. The format of LIST is as follows:

```
LIST <list name> ;

<expression> , <expression> , ... <expression> ;
<expression> , <expression> , ... <expression> ;
<expression> , <expression> , ... <expression> ;

END_LIST ;
```

The expression lists can then be referenced by other statements described in this section.

5.8.2 SEARCH Statement

SEARCH locates a specific entry in a LIST. SEARCH has the following format:

```
SEARCH <list name> <expression> , ... , <expression>
      GET <variable> , ... <variable>
      FLAG <variable> ;
```

The specified list is searched sequentially until an entry is found where each of the SEARCH expressions is equal in value to the corresponding LIST expression in the same expression list. The variables in the GET clause are then set to the corresponding remaining expressions of the expression list, and the FLAG variable is set to TRUE. If no match is found, the variables specified in the GET clause are unchanged, and the FLAG variable is set to FALSE.

5.9 SIGNALLING

This section describes the signalling facilities of CFL.

5.9.1 Signalling

Signalling breaks the control flow in the report to handle special conditions. Control goes to a special routine established by the user called a handler routine. When a condition is signalled by the user using the signalling statements, the most recently declared handler routine is called. The handler routine can then take the appropriate action.

Any routine can establish a handler routine. When a condition is signalled by the user, the user can optionally suppress the change in the flow of control, and cause the handler to return to the routine executing the signal.

When a condition is signalled, a message describing the event is appended to the file ERROR, if the file exists. The message inserted in the ERROR file consists of a sequence of comma-delimited quoted strings, corresponding to the arguments to the SIGNAL-class statements.

5.9.2 ENABLE Statement

ENABLE has the following format:

```
ENABLE [ MODULE <expression> ] PROCEDURE <expression> ;
```

The procedure becomes the condition handler for this procedure and all called procedures, unless a called procedure in turn has an ENABLE.

5.9.3 SIGNAL Statement

SIGNAL has the following format:

```
SIGNAL <message code> PARAMETERS <expression list> ;
```


The message code and expressions in the expression list are ASCII strings. When the SIGNAL statement is executed these expressions are evaluated and the resulting ASCII strings are appended to the ERROR file as quoted strings separated by commas, as follows:

```
'<message code>', '<expression1>', '<expression2>' ..., '<expressionN>'
```

The signal-handling routine is then called. After execution of the signal-handling procedure, execution resumes following the statement.

5.9.4 SIGNAL_STOP Statement

The SIGNAL_STOP statement is the same as the SIGNAL statement, except that after execution of the signal-handling procedure, execution resumes following the call to the procedure that executed the ENABLE statement.

```
SIGNAL_STOP <message code> PARAMETERS <expression list> ;
```

5.9.5 MESSAGE Statement

MESSAGE has the same format as SIGNAL. It causes the appended string to be placed in the ERROR file, but does not cause any signal processing.

```
MESSAGE <message code> PARAMETERS <expression list> ;
```

5.9.6 CRASH Statement

CRASH causes an immediate abort of RPT. Use it in cases of error handling where the signalling mechanism is inadequate. The CRASH statement has the following format:

```
CRASH ;
```

CRASH causes a detailed dump of many of RPT's internal data structures.

5.10 PRINT FORMATTING

This section describes the output-formatting facilities of CFL.

5.10.1 FORMAT keyword string

The FORMAT keyword on the WRITE and WRITE GROUP statements expresses output-formatting information. The keyword has the following syntax:

```
FORMAT <format string>
```

The format string can be any ASCII string. The string is output, after substitution specified by output directives. The directives have the format

```
!n(mdd)
```

where dd is the two-character directive, m is the optional argument, and n is the optional repeat count. The parentheses need not be included if there is no repeat count. This syntax is the same as that for the VMS %FAO facility. A double-exclamation mark, !!, prints as a single exclamation mark.

Multiple format strings can be specified one after the other separated by the concatenation operator, the vertical bar (|, ASCII 174). They are treated as one concatenated string.

The allowable directives are specified in the following sections.

5.10.1.1 Control Directives - The control directives control the processing of the format string. Here are the control directives:

!nCE Repeat the FORMAT clause.

!nCF When used with a WRITE statement, this directive terminates output if the values of all expressions have been output. When used with a WRITE GROUP statement, this directive terminates output if all fields in the specified group have been output. The effect in both cases is to terminate evaluation and output of the format string if there are no more values to be output.

5.10.1.2 Formatting Directives - The formatting directives output carriage-control information. Here are the formatting directives:

!nFC Print following output beginning at column n.

!nFS Space the current output print column forward n columns.

!nFL Output n-1 blank lines. Printing resumes on the line following the blank lines. The default, n=1, causes output to begin on the line following the current line.

!nFP Output a page break.

5.10.1.3 Data-formatting Directives - The data-formatting directives control the output of data. Here are the data-formatting directives:

!nDF Print the field name of the current output field. The argument specifies the field width to be used for the name. The name is printed left-justified.

!nDP Print the current output field.

The argument specifies the field width to be used for the field. For numeric fields, the field width n must be greater than the field width specified when the field was defined. If all fields have been printed, output terminates.

5.11 USER INTERFACE HANDLING

This section describes the user interface to the compiler.

5.11.1 Overview of User Interface Handling

The compiler implements two user interface modes, command mode and OPTION mode. In OPTION mode, the compiler requests a command line, followed by requests for OPTIONS, which are terminated when a line beginning with a slash (/) is entered, at which point the compilation takes place.

5.11.2 Command Mode

In command mode, the compiler requires a command line of the following syntax:

```
<output>,<listing>,<symbols>=<input>,<symbols>
```

All files on the left of the equals (=) character are output files, and all files to the right of the = character are input files.

The output file is the compiled module output file. It has the default file type .ICF.

The listing file is the compilation listing file. It has the default file type .LST.

The symbols output file is the compilation symbol table, which must be used as input to any compilation of a module to be called from this module at execution time. It has the default file type .SYM.

The input file is the .CNF source file.

The symbols input file is a compilation symbol table from the module which calls the module being compiled at execution time.

5.11.3 Option Mode

In option mode, the compiler accepts the following option:

- LITERAL

The LITERAL option has the form:

```
LITERAL group.name = value
```

This is the same syntax as for the LITERAL statement in the source. The LITERAL is declared for the duration of the compilation. The only valid values are single items: quoted strings, numeric values, and logical TRUE and FALSE values. Numeric values must be positive decimal values that are treated as machine values.

5.12 ERLCFL REPORT MESSAGES

ERLCFL-F-ASCIIIBIG, ASCII literal quoted string too long for type.

Explanation: An ASCII radix numeric literal in a control file source module contains too many characters for the specified numeric type.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BADDIGIT, Invalid numeric digit in conversion.

Explanation: A numeric literal or the ASCII string argument for the %COD\$OCTAL, %COD\$DECIMAL, %COD\$HEX, %COD\$BCD, %COD\$BINARY, or %COD\$MACHINE function contains an illegal character for the specified radix or was null or blank.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BITFLDSIZ, Bit or field too large in extraction operation.

Explanation: The bit or field in an extraction operation exceeds the size of the value on which the extraction is performed.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BITNOPREC, A BIT or FIELD must have a preceding data item.

Explanation: A BIT or FIELD declaration in a control file source module must be preceded by a data item within the declaration.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BITNOTVAR, A BIT or FIELD not allowed on variable-length data item.

Explanation: A BIT or FIELD declaration in a control file source module is not allowed on a variable length data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BITNUMINV, BIT number outside the declared data item.

Explanation: The bit number in a BIT declaration for a data item is too large for the data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-BITTOOHIG, Bit number too large for specified storage unit.

Explanation: The bit number specified by the character string portion of a #BI, #WI, #LI, #QI, or #VI numeric literal is too large for the specified value size.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-BYTERAD50, A BYTE data item cannot print in RAD50.

Explanation: The print radix for a BYTE declaration can not be RAD50.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-CASENOTDAT, A declaration clause must be in a data declaration.

Explanation: A declaration CASE clause attempted to declare data but was not contained within a declaration.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-CFLINPUT, Could not open input source file.

Explanation: CFL could not open the input source file specified on the command line.

User Action: Check the file specification, and make sure that you have access to the specified file.

ERLCFL-F-CFLISTING, Could not create listing output file.

Explanation: CFL could not create the listing output file specified on the command line.

User Action: Check the file specification, and make sure that you have access to the specified file.

ERLCFL-F-CFLMODULE, Could not create module output file.

Explanation: CFL could not create the module output file specified on the command line.

User Action: Check the file specification, and make sure that you have access to the specified file.

ERLCFL-F-CFLSYMBOL, Could not open symbol file for input.

Explanation: CFL could not open the input symbol file specified on the command line.

User Action: Check the file specification, and make sure that you have access to the specified file.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-CFLSYMOUT, Could not create symbol output file.

Explanation: CFL could not create the output symbol file specified on the command line.

User Action: Check the file specification, and make sure that you have access to the specified file.

ERLCFL-F-CMDOPTERR, Option line syntax error.

Explanation: CFL encountered a syntax error on the option line input.

User Action: If the error occurred in a DIGITAL-supplied command file please submit an SPR, otherwise correct the error and run CFL again.

ERLCFL-F-CMDSPCERR, Command line syntax error.

Explanation: CFL encountered a syntax error on the command line input.

User Action: If the error occurred in a DIGITAL-supplied command file please submit an SPR, otherwise correct the error and run CFL again.

ERLCFL-F-DECTOOBIG, Declaration too large, too many symbols.

Explanation: A declaration in a control file is too large to be compiled.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-DIVZERO, Attempt to divide by zero.

Explanation: A control file module contains a division by zero in a compile-time constant expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-EXPTOOBIG, Operator stack overflow. Expression too complex.

Explanation: An expression in a control file source module is too complex to be compiled.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-EXPTOOLAR, Operator stack overflow. Expression too complex.

Explanation: An expression in a control file source module is too complex to be compiled.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-FIELDBIG, FIELD exceeds size of the declared data item.

Explanation: A FIELD declaration in a control file source module exceeds the bounds of its corresponding data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FIELDBITI, FIELD starting bit is outside the declared data item.

Explanation: A FIELD declaration in a control file source module exceeds the bounds of its corresponding data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FIELDSMAL, FIELD width must be at least one bit.

Explanation: A FIELD declaration in a control file source module did not have a width specified.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FILERCLOS, File close error.

Explanation: An error occurred when CFL attempted to close a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLCFL-F-FILERREAD, File read error.

Explanation: An error occurred when CFL attempted to read a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLCFL-F-FILERWRIT, File write error.

Explanation: An error occurred when CFL attempted to write to a file.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLCFL-F-FILINTOPN, Internal error - file already open.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-FILINVCOD, Internal error - invalid file code for specified operation.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-FLUSHINV, FLUSH attribute not allowed with KEEP attribute.

Explanation: A control file source module had both the FLUSH and KEEP module attributes specified.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNFIELDS, Invalid conversion code argument to time conversion function.

Explanation: A control file module contains a time conversion function with an illegal value for the conversion code argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNINVPOI, Invalid string pointer value in string function.

Explanation: A control file module contains a %STR\$PARSE or %STR\$QUOTE function where the value of the pointer argument is larger than the length of the string argument.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNNOTCHA, Argument to STR\$CHAR is not in valid range for character.

Explanation: The value of the argument for the %STR\$CHAR function must be in the range 0 to 127(10).

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNNOTCOM, Function call not allowed in compile-time constant expression.

Explanation: A control file module contains a function call that could not be evaluated at compile-time, where a compile-time constant expression was required.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNQUOODD, Quote string in STR\$QUOTE function must have even length.

Explanation: A control file module contains a %STR\$QUOTE function, where the quote string argument is not an even length.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-FUNSTRSIZ, Output string from string function too large.

Explanation: A control file module executed a string function which resulted in a string longer than 255 characters.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNWRONGA, Incorrect number of arguments in function call.

Explanation: A control file source module contains a function call with the wrong number of arguments.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-FUNWRONGC, Incorrect number of arguments in function call.

Explanation: A control file source module contains a function call with the wrong number of arguments.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-HEAPOVERF, Heap too small to hold value. Overflow.

Explanation: The heap used for processing values and expressions has overflowed.

User Action: Edit CFLBLD.CMD to increase the psect extension for psect VHEAP0, and rebuild CFL.

ERLCFL-F-IFNOTDATA, A declaration IF clause must be in a data declaration.

Explanation: An IF clause cannot be used to declare data outside of a declaration in a control file source module.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-ILLCHAR, Illegal character in input.

Explanation: An illegal character was found in a control file source module.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-INTEOTMAN, Internal - More than one operator on stack at term end.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-INTEOTNUL, Internal - End of term reached with null operator stack.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTEXPNOO, Internal - Operator missing from operator stack.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTFUNEND, Internal - Stack entry missing at function termination.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTFUNMIS, Internal - Function code missing from operator stack.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTFUNNOT, Internal - Function code missing at function termination.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTOPRNOT, Internal - Operator outside of an expression term.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTPROUND, Internal - Compiler internal production stack underflow.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INTSYMLNK, Internal error - Invalid symbol linkage setup in module.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-INTWRONGP, Internal - Wrong production popped internal production.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INVFUNCT, Invalid function name specified.

Explanation: A control file source module specified an invalid function name.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-INVNUMSIZ, Internal - A numeric variable has an invalid size.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-INVPOIACT, Invalid POINTER-statement action name.

Explanation: A control file source module specified an invalid action for a POINTER statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-INVRADCNV, Internal error - Invalid radix code for conversion.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-KEEPINV, KEEP attribute not allowed with FLUSH attribute.

Explanation: A control file source module had both the FLUSH and KEEP module attributes specified.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-LITINVTYP, Internal error - Literal in literal table has invalid type.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-LITNOVALU, Internal error - no value to load into literal value.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-MODATTRIN, Invalid module attribute name specified.

Explanation: A control file source module specified an invalid module attribute.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-MODZERO, Attempt to modulus by zero.

Explanation: A control file source module contains a modulus by zero in a compile-time constant expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-NOQUOTE, String literal missing closing quote.

Explanation: A string literal in a control file source module was not terminated by a closing quote.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-NULLOPERA, Internal - Null suffix operand on non-suffix operator.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-NUMFILLCH, A print fill character string must contain one character.

Explanation: A print fill character in a declaration in a control file source module must contain one character.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-NUMINVOPR, Invalid numeric double-operand operation code.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-OPRFLSCOM, Internal error - attempt to flush a CTCE operand.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-OPRINVLOG, Attempt to perform logical operation on an invalid type.

Explanation: A control file source module attempted to perform a logical operation with operands that were neither NUMERIC nor LOGICAL.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-OPRNOTCOM, Operator in CTCE cannot be evaluated at compile-time.

Explanation: A compile-time constant expression in a control file source module contains an operator which could not be evaluated at compile-time.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-OPRNOTIMP, Operation not implemented.

Explanation: A control file source module attempted to perform a multiplication where both operands were larger than a word value.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-PARSEREOF, Premature EOF encountered.

Explanation: The end-of-file was reached on a control file source module, before the end of the module was reached.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-PROSTKOV, Compiler internal production stack overflow.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-PRSSTKOV, Parse stack overflow.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-RAD50BYTE, Cannot convert a byte using RAD50 conversion.

Explanation: A control file source module attempted to convert an ASCII string or numeric literal to a BYTE using RAD50 conversion.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-RADINVCOD, Invalid radix code string in radix literal.

Explanation: A control file source module contains an invalid radix code in a numeric literal.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-RADINVRAD, Invalid radix character specified in radix literal.

Explanation: A control file source module contains an invalid radix code in a numeric literal.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-RADLITINV, Invalid literal type character in radix literal.

Explanation: A control file source module contains an invalid type character in the radix portion of a numeric literal.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-RELINVCOD, Invalid relational operator.

Explanation: This is an internal error within CFL.

User Action: Please submit an SPR with any information you have.

ERLCFL-F-RESBITILL, A BIT or FIELD data item cannot be declared RESERVED.

Explanation: A control file source module contains a BIT or FIELD declaration for a RESERVED data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-RESINVPRT, Print expression list not allowed on RESERVED data.

Explanation: A control file source module contains a print expression for a RESERVED data item.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-SPCNOTVAR, Special variable cannot be used in a CTCE.

Explanation: A control file source module used one of the predeclared special variables in a compile-time constant expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-SRCDATERR, Failed to get creation date of source file

Explanation: An error occurred when CFL tried to get the creation date of the control file source module.

User Action: Check for file access conflicts, device errors, or low pool condition.

ERLCFL-F-SUBEXTBIG, Substring extraction end element exceeds string.

Explanation: A control file module attempted to perform a substring extraction in which the substring exceeded the end of the string.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-SYMNOTCOM, A variable is not valid in a compile-time constant expression.

Explanation: A control file source module contains a variable in a compile-time constant expression.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-SYMNOTLIT, Specified LITERAL symbol name not part of LITERAL group.

Explanation: A control file module contains a reference to a LITERAL symbol which has not been defined for the specified LITERAL group.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-SYNTAXERR, Syntax error.

Explanation: CFL encountered a syntax error while compiling the control file source module.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-TABLEBIG, TABLE element has too many literal values.

Explanation: A control file source module contains a TABLE element with too many values for the corresponding TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-TABLESMAL, TABLE element has too few literal values.

Explanation: A control file source module contains a TABLE element with too few values for the corresponding TABLE.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

CONTROL FILE LANGUAGE GUIDE

ERLCFL-F-VALSTKOV, Value stack overflow.

Explanation: The stack used for processing values and expressions has overflowed.

User Action: Edit CFLBLD.CMD to increase the psect extension for psect VLSTK0, and rebuild CFL.

ERLCFL-F-VALUESIZE, Value in expression is too large.

Explanation: A control file source module contains a compile-time constant expression in which an intermediate value or the final value is too large.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-VALUETYPE, Value in expression is wrong type.

Explanation: A control file source module contains a compile-time constant expression in which an intermediate value or the final value is the wrong type.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-VARLITGRP, A variable name cannot have the same group name as a LITERAL.

Explanation: A control file module contains a declaration with the same group name as a LITERAL declaration.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

ERLCFL-F-WRITEDES, WRITE-class statement has invalid destination.

Explanation: A control file module contains an invalid destination for the TO clause of a WRITE or WRITE_GROUP statement.

User Action: Correct the user-written module or submit an SPR for DIGITAL-supplied modules.

APPENDIX A

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

When RPT starts operation, it scans ERRLOG.ULB and generates descriptors in its work space for all the modules it finds in the library. RPT searches these descriptors for the appropriate modules when it generates reports. Therefore, if you remove modules for devices that your system does not have, RPT operation is faster and does not require as much memory or disk space.

An indirect command file, TUNE.CMD, is included with your Error Logging System. The command file is in the universal library LB:[1,6]ERRLOGETC.ULB. This command file allows you to remove unneeded modules from ERRLOG.ULB.

To use TUNE.CMD you must first extract it from the library, using the following LBR command:

```
LBR TUNE.CMD=LB:[1,6]ERRLOGETC.ULB/EX:TUNE (RET)
```

The file, TUNE.CMD, is copied to your current directory and you can execute the command.

The file prompts you for a processor type and for a list of the devices you want to use with Error Logging. It then removes the devices you do not select from ERRLOG.ULB and writes a new ERRLOG.ULB to the current UFD.

TUNE.CMD requires the actual physical names of the devices you select (RL01, RL02, for example). Since some modules in ERRLOG.ULB handle more than one physical device type, the tune command only includes the module once. If you select a device name more than once, the command displays the following message on your terminal:

Module <module name> replaced.

The message is only for your information. The command file then continues to include the other modules you specified.

Example A-1 is an example of use of the RSX-11M version of TUNE.CMD. Files for other systems are similar. This file creates a library for an 11/70 with:

Devices

TU45	RK05
RP06	RK06
RP05	RK07
RM03	TU56
RM05	RX02
RM80	TU58
RS04	

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Example A-1 Sample Execution of TUNE.CMD

@LB:[1,6]TUNE (RET)

```
;
;      This command procedure is used to tune the Error Log Control file for
;      your system configuration. The procedure prompts for the location of
;      the master control file (the shipped file), the CPU type, and the
;      error logging devices available on your system. It then creates a new
;      version of the control file that contains only the required support.
;      The original control file is left unchanged.
```

```
;
;      All files are created in your default directory on the default
;      device. When cleaning up, all files with the extension of .ICF as
;      well as TEMPTUNE0.TMP;* and TEMPTUNE1.TMP;* are deleted.
```

```
* Continue? [Y/N]: Y (RET)
```

```
* Location of master file [D: LB:[1,6]ERRLOG.ULB] [S]: (RET)
```

```
;
;      Now enter the CPU type. Hit the escape key for a list of legal
;      CPU types.
```

```
* Enter CPU type [S]: (ESC)
```

```
;
;      The acceptable CPU types are:
```

```
;      11/03, 11/04, 11/05, 11/10
;      11/20, 11/23, 11/24, 11/34
;      11/35, 11/40, 11/44, 11/45
;      11/50, 11/55, 11/60, 11/70
;      11/74
```

```
* Enter CPU type [S]: 11/70 (RET)
```

```
;
;      Now enter the devices in your configuration separated by commas.
;      Terminate by entering a period. Hit the escape key for a list of
;      acceptable device names.
```

```
* Enter device name(s) [S]: (ESC)
```

```
;
;      Below is a list of acceptable device names. If you have more than
;      one type of device listed as "x or y or z" you need enter only one.
;      For example, if you have RP04s and RP06s you need only enter RP04
;      or RP06 - not both.
```

```
;
;      The acceptable device names are:
```

```
;      TU56 (DECTape)
;      TU58 (DECTape II)
;      TU60 (Cassette)
;      RP04 or RP05 or RP06
;      RP07
;      RS11
;      RK03 or RK05
;      RL01 or RL02
;      RK06 or RK07
```

(continued on next page)

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Example A-1 (Cont.) Sample Execution of TUNE.CMD

```

;      RP02 or RP03
;      RM02 or RM03
;      RM05
;      RM80
;      RS03 or RS04
;      RA60
;      RA80/RA81
;      RC25 or RD51 or RX50
;      RX01
;      RX02
;      ML11
;      TE16 or TU16 or TU45
;      TU77
;      TE10 or TU10 or TS03
;      TU78
;      TS11 or TU80
;      TSV05
* Enter device name(s) [S]: TU45,RP06,RP05,RM03,RM05,RM80 (RET)
* Enter device name(s) [S]: RS04,RK05,RK06,RK07,TU56,RX02,TU58 (RET)
* Enter device name(s) [S]: . (RET)
;
;      Extract the files from the master library.
;
LBR @TEMPTUNE0.TMP
;
;      Build the new library. Note that you may see messages like "Module
;      "XYZZY" replaced" if you have selected more than one device having
;      the same mnemonic. For example, selecting RK06 and RK07 support will
;      produce this message. This type of message can be ignored.
;
LBR @TEMPTUNE1.TMP
Module "ERP456" replaced
Module "ERK67 " replaced
Module "NRK67 " replaced

;      Now clean up.
;
PIP TEMPLIB.ULB/TR
PIP ERRLOG.ULB/RE/NV=TEMPLIB.ULB
PIP *.ICF;*,TEMPTUNE0.TMP;*,TEMPTUNE1.TMP;*/DE
;
;      Finished.
;

```

You can then copy the new library to [1,6] where it will become the default library for RPT. You should maintain the original DIGITAL-supplied ERRLOG.ULB, either on LB:[1,6] or in another location. You can then use TUNE.CMD again later, on the original ULB, to add support for devices that you have taken out.

To list the modules in the library, use the following LBR command:

```
>LBR ERRLOG.ULB/LI (RET)
```

Table A-1 lists the modules in the DIGITAL-supplied library for RSX-11M.

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Table A-1
Modules in ERRLOG.ULB for RSX-11M

Module Name	Module Description
DEVSM1	Defines tables for device modules
DEVUDA	MSCP related module
DISPAT	Entry module for error log control file
DMPALL	Processes errors from unknown events
DSP1M1	Processes Error Logger Command Packets
DSP2M1	Processes device error packets
DSP3M1	Processes device information packets
DSP4M1	Processes device control information
DSP5M1	Processes CPU/memory detected errors
DSP6M1	Processes system control information
DSP7M1	Processes control information packets
EML11	Control File Device Modules
ERK05	
ERK67	
ERL12	
ERM05	
ERM23	
ERM80	
ERP07	
ERP23	
ERP456	
ERS11	
ERS34	
ERX01	
ERX02	
ETA11	
ETC11	
ETS11	
ETU58	
ETU77	
ET0310	
ET1645	
ETSV05	
ERRORM	Processes control file error conditions
EUNKWN	Processes errors from unknown devices
E11XX	Processes CPU/memory packets
E1134	
E1144	
E1160	
E117X	
FINLM1	Cleans up control file after processing
FM1NM1	Prints narrow or wide width reports
FM1WM1	
FM2CM1	
FM3CM1	
FM4NM1	
FM4WM1	

(continued on next page)

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Table A-1 (Cont.)
Modules in ERRLOG.ULB for RSX-11M

Module Name	Module Description
INITM1	Initializes control file modules
MSCPCE	MSCP Device Modules
MSCPAT	
MSCP80	
MSCP60	
MSCPEN	
MSCPTO	
MSCPSD	
NML11	Generates notes from device modules
NRK67	
NRM05	
NRM23	
NT0310	
NTS11	
PARSEM	RPT command line parser
PARS1M	
PARS2M	
PARS3M	
PRS2AM	
PRS2BM	
SELTM1	Selects packets to process

Table A-2 lists the modules in the DIGITAL-supplied library for RSX-11M-PLUS.

Table A-2
Modules in ERRLOG.ULB for RSX-11M-PLUS

Module Name	Module Description
DEVSM1	Defines tables for device modules
DEVUDA	MSCP related module
DISPAT	Entry module for error log control file
DMPALL	Processes errors from unknown events
DSP1P1	Processes Error Logger Command Packets
DSP2P1	Processes device error packets
DSP3P1	Processes device information packets
DSP4P1	Processes device control information
DSP5P1	Processes CPU/memory detected errors
DSP6P1	Processes system control information
DSP7P1	Processes control information packets

(continued on next page)

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Table A-2 (Cont.)
Modules in ERRLOG.ULB for RSX-11M-PLUS

Module Name	Module Description
EML11	Control File Device Modules
ERK05	
ERK67	
ERL12	
ERM05	
ERM23	
ERM80	
ERP07	
ERP23	
ERP456	
ERS11	
ERS34	
ERX01	
ERX02	
ETA11	
ETC11	
ETS11	
ETU58	
ETU77	
ET0310	
ET1645	
ETSV05	
ERRORM	Processes control file error conditions
EUNKWN	Processes errors from unknown devices
E11XX	Processes CPU/memory packets
E1144	
E117X	
FINLP1	Cleans up control file after processing
FMTNP1	Prints narrow width reports
FMTWP1	Prints wide width reports
INITM1	Initializes control file modules
MSCPCE	MSCP Device Modules
MSCPAT	
MSCP80	
MSCP60	
MSCPEN	
MSCPTO	
MSCPSD	
NML11	Generates notes from device modules
NRK67	
NRM05	
NRM23	
NT0310	
NTS11	

(continued on next page)

TUNING THE ERROR LOGGING UNIVERSAL LIBRARY

Table A-2 (Cont.)
Modules in ERRLOG.ULB for RSX-11M-PLUS

Module Name	Module Description
PARSEM PAR1M PAR2M PAR3M PRS2AM PRS2BM	RPT command line parser
SELTm1	Selects packets to process
SMRYEP	Prints Error Summary Report
SMRYGP	Prints Geometry Summary Report
SMRYHP	Prints History Summary Report

APPENDIX B

DRIVE SERIAL NUMBERS

RPT reports drive serial numbers for those devices that have serial numbers. Table B-1 lists the drives that provide serial numbers and the significant digits RPT uses from those serial numbers. The number of digits varies by drive type. They appear in binary coded decimal (BCD) format.

RPT obtains the serial number from the device electronics. The number is selected through a series of jumpers within the device that are set at manufacturing time. These jumpers match the low order digits of the actual device serial number. If any of these jumpers is altered, the drive will have a different serial number, unless the new jumpers are set to reflect the actual device serial number. Therefore, the number RPT prints may not be the same as the number on the device identification plate.

Table B-1
Significant Digits in Drive Serial Numbers

Device/Controller	Significant Digits in Serial Number
RK06	12 bits, 3 BCD digits
RK07	12 bits, 3 BCD digits
RM80	16 bits, 4 BCD digits
TU77	16 bits, 4 BCD digits
TU78	16 bits, 4 BCD digits
RP04	16 bits, 4 BCD digits
RP05	16 bits, 4 BCD digits
RP06	16 bits, 4 BCD digits
RP07	16 bits, 4 BCD digits
RX01	N/A
RX02	N/A
RM02	16 bits, 4 BCD digits
RM03	16 bits, 4 BCD digits
RM05	16 bits, 4 BCD digits
TA11	N/A
RS11	N/A
RP02	N/A
RP03	N/A
ML11	16 bits, 4 BCD digits
TE16/RH11/RH70	16 bits, 4 BCD digits
TU16/RH11/RH70	16 bits, 4 BCD digits
TU45	16 bits, 4 BCD digits
TS03	N/A
TE10/TMB11	N/A
TU10/TMB11	N/A

(continued on next page)

DRIVE SERIAL NUMBERS

Table B-1 (Cont.)
Significant Digits in Drive Serial Numbers

Device/Controller	Significant Digits in Serial Number
RS03	N/A
RS04	N/A
TS11	N/A
TC11	N/A
RL01	N/A
RL02	N/A
TSV05	N/A
RA60/RA80/RA81	6 digits
RC25/RD51/RX50	6 digits
TU80	N/A

APPENDIX C

ERROR LOG PACKET FORMAT

Example C-1 shows the format of an error log packet in memory, as described in the system macro EPKDF.MAC.

When a device error is logged, the error log packet contains device supplied information to describe the error. This information usually consists of the device registers and some additional information supplied by the system.

Error logging also writes context information into the error log packet. This information includes the time and date of the error, information about the system that logged the error, and information about the I/O operation that generated the error.

The error logging system also creates packets for events in the system that are not errors, but are important to the interpretation of errors, such as the time error logging starts or stops.

Example C-1 Error Log Packet Format

```
.IIF NDF S$$YDF ,      .NLIST

;
;      COPYRIGHT (c) 1983 BY
;      DIGITAL EQUIPMENT CORPORATION, MAYNARD,
;      MASSACHUSETTS.  ALL RIGHTS RESERVED.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
; ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
; COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
; OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
; TRANSFERRED.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
; AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
; CORPORATION.
;
; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
; SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
;
;      IDENT /1.03/
;
; Modified By:
;
;      C. PUTNAM      - 03-FEB-82      1.01
;                    - Add M-PLUS Definition of 'NUX' (No UCB extension) bit
;
```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```
;
; C. PUTNAM      - 13-SEP-82      1.02
;               - Add Q-BUS flag to HEADER subpacket
;
; C. PUTNAM      - 06-JAN-83      1.03
;               - Add E$STMS (recovered timeout) device error
;               - Now E$STMO means unrecovered timeout device error
;
```

```
.MACRO EPKDF$,L,B
```

```
;+
; Error Message Block Definitions
```

```
;-
```

```
.ASECT
```

```
; Header Subpacket
```

```
;
;
; +-----+-----+
; | Subpacket Length in Bytes |
; +-----+-----+
; | Subpacket Flags |
; +-----+-----+
; | Format Identification | Operating System Code |
; +-----+-----+
; | Operating System Identification |
; +-----+-----+
; | Flags | Context Code |
; +-----+-----+
; | Entry Sequence |
; +-----+-----+
; | Error Sequence |
; +-----+-----+
; | Entry Type Subcode | Entry Type Code |
; +-----+-----+
; | Time Stamp |
; +-----+-----+
; | Reserved | Processor Type |
; +-----+-----+
; | Processor Identification (URM) |
; +-----+-----+
```

```
.=0
```

```
E$HLGH:'L'      .BLKW 1      ; Subpacket length in bytes
E$HSBF:'L'      .BLKW 1      ; Subpacket Flags
E$HSYS:'L'      .BLKB 1      ; Operating System Code
E$HIDN:'L'      .BLKB 1      ; Format Identification
E$HSID:'L'      .BLKB 4      ; Operating System Identification
E$HCTX:'L'      .BLKB 1      ; Context Code
E$HFLG:'L'      .BLKB 1      ; Flags
E$HENS:'L'      .BLKW 1      ; Entry Sequence Number
E$HERS:'L'      .BLKW 1      ; Error Sequence Number
E$HENC:'L'      .BLKB 1      ; Entry Code
E$HTYC:'L'      .BLKB 1      ; Entry Type Code
E$HTYS:'L'      .BLKB 1      ; Entry Type Subcode
E$HTIM:'L'      .BLKB 6      ; Time Stamp
E$HPTY:'L'      .BLKB 1      ; Processor Type
                .BLKB 1      ; Reserved
```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```

E$HURM: 'L'      .BLKW  1      ; Processor Identification (URM)

                        .EVEN
E$HLEN: 'L'      ; Length

;
; Subpacket Flags for E$HSBF
;
SM.ERR  = 'B'      1 ; Error Packet
SM.HDR  = 'B'      1 ; Header Subpacket
SM.TSK  = 'B'      2 ; Task Subpacket
SM.DID  = 'B'      4 ; Device Identification Subpacket
SM.DOP  = 'B'     10 ; Device Operation Subpacket
SM.DAC  = 'B'     20 ; Device Activity Subpacket
SM.DAT  = 'B'     40 ; Data Subpacket
SM.MBC  = 'B'    20000 ; 22-bit massbus controller present
SM.CMD  = 'B'    40000 ; Error Log Command Packet
SM.ZER  = 'B'   100000 ; Zero I/O Counts

;
; Codes for field E$HIDN
;
EH$FOR  = 'B'      1 ; Current packet format

;
; Flags for the error log flags byte ($ERFLA) in the exec.
;
ES.INI  = 'B'      1 ; Error log initialized
ES.DAT  = 'B'      2 ; Error log receiving data packets
ES.LIM  = 'B'      4 ; Error limiting enabled
ES.LOG  = 'B'     10 ; Error logging enabled

;
; Type and Subtype Codes for fields E$HTYC and E$HTYS
;
; Symbols with names E$Cxxx are type codes for field E$HTYC,
; symbols with names E$Sxxx are subtype codes for field E$HTYS.
;
E$CCMD  = 'B'      1 ; Error Log Control
E$SSTA  = 'B'      1 ; Error Log Status Change
E$SSWI  = 'B'      2 ; Switch Logging Files
E$SAPP  = 'B'      3 ; Append File
E$SBAC  = 'B'      4 ; Declare Backup File
E$SSHO  = 'B'      5 ; Show
E$SCHL  = 'B'      6 ; Change Limits

E$CERR  = 'B'      2 ; Device Errors
E$SDVH  = 'B'      1 ; Device Hard Error
E$SDVS  = 'B'      2 ; Device Soft Error
E$STMO  = 'B'      3 ; Device Interrupt Timeout (HARD)
E$SUNS  = 'B'      4 ; Device Unsolicited Interrupt
E$STMS  = 'B'      5 ; Device Interrupt Timeout (SOFT)

E$CDVI  = 'B'      3 ; Device Information
E$SDVI  = 'B'      1 ; Device Information Message

E$CDCI  = 'B'      4 ; Device Control Information
E$SMOU  = 'B'      1 ; Device Mount
E$SDMO  = 'B'      2 ; Device Dismount
E$SRES  = 'B'      3 ; Device Count Reset
E$SRCT  = 'B'      4 ; Block Replacement
E$CCPU  = 'B'      5 ; CPU Detected Errors
E$SMEM  = 'B'      1 ; Memory Error

```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```

E$SINT  ='B'          2 ;      Unexpected Interrupt

E$CSYS  ='B'          6 ; System Control Information
E$SPWR  ='B'          1 ;      Power Recovery

E$CCTL  ='B'          7 ; Control Information
E$STIM  ='B'          1 ;      Time Change
E$SCRS  ='B'          2 ;      System Crash
E$SLOA  ='B'          3 ;      Device Driver Load
E$SUNL  ='B'          4 ;      Device Driver Unload
E$SHRC  ='B'          5 ;      Reconfiguration Status Change
E$SMES  ='B'          6 ;      Message

E$CSDE  ='B'         10 ; Software Detected Events
E$SABO  ='B'          1 ;      Task Abort

;
; Codes for Context Code entry E$HCTX
;
EH$NOR  ='B'          1 ; Normal Entry
EH$STA  ='B'          2 ; Start Entry
EH$CRS  ='B'          3 ; Crash Entry

;
; Codes for Flags entry E$HFLG
;
EH$VIR  ='B'          1 ; Addresses are virtual
EH$EXT  ='B'          2 ; Addresses are extended
EH$COU  ='B'          4 ; Error counts supplied
EH$QBS  ='B'         10 ; Q-BUS CPU

;
; Task Subpacket
;
+-----+
| Task Subpacket Length |
+-----+
| Task Name in RAD50    |
+-----+
| Task UIC              |
+-----+
| Task TI: Device Name  |
+-----+
| Flags                  | Task TI: Unit Number |
+-----+

.=0

E$TLGH:'L'      .BLKW  1      ; Task Subpacket Length
E$TTSK:'L'      .BLKW  2      ; Task Name in RAD50
E$TUIC:'L'      .BLKW  1      ; Task UIC
E$TTID:'L'      .BLKB  2      ; Task TI: Device Name
E$TTIU:'L'      .BLKB  1      ; Task TI: Unit
E$TFLG:'L'      .BLKB  1      ; Flags

.EVEN
E$TLEN:'L'

;
; Flags for entry E$TFLG
;
ET$PRV  ='B'          1 ; Task is Privileged

```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

ET\$PRI ='B' 2 ; Terminal is Privileged

; Device Identification Subpacket

;	+	-----		+
;		Device Identification Subpacket Length		
;	+	-----		+
;		Device Mnemonic Name		
;	+	-----		+
;		Controller Number	Device Unit Number	
;	+	-----		+
;		Physical Subunit #	Physical Unit #	
;	+	-----		+
;		Physical Device Mnemonic (RSX-11M-PLUS only)		
;	+	-----		+
;		Reserved	Flags	
;	+	-----		+
;		Volume Name of Mounted Volume		
;				
;				
;				
;				
;	+	-----		+
;		Pack Identification		
;	+	-----		+
;		Device Type Class		
;	+	-----		+
;		Device Type		
;	+	-----		+
;		I/O Operation Count Longword		
;	+	-----		+
;		Hard Error Count	Soft Error Count	
;	+	-----		+
;		Blocks Transferred Count (RSX-11M-PLUS only)		
;	+	-----		+
;		Cylinders Crossed Count (RSX-11M-PLUS only)		
;	+	-----		+

. = 0

E\$ILGH: 'L'	.BLKW	1	; Device Identification Subpacket Length
E\$ILDV: 'L'	.BLKW	1	; Device Mnemonic Name
E\$ILUN: 'L'	.BLKB	1	; Device Unit Number
E\$IPCO: 'L'	.BLKB	1	; Controller Number
E\$IPUN: 'L'	.BLKB	1	; Physical Unit Number
E\$IPSU: 'L'	.BLKB	1	; Physical Subunit Number

.IF DF R\$MPL

E\$IPDV: 'L'	.BLKW	1	; Physical Device Mnemonic
--------------	-------	---	----------------------------

	.ENDC ; R\$MPL		
E\$IFLG: 'L'	.BLKB	1	; Flags
	.BLKB	1	; Reserved
E\$IVOL: 'L'	.BLKB	12.	; Volume Name

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```

E$IPAK:'L'      .BLKB    4      ; Pack Identification
E$IDEV:'L'      ; Device Type
E$IDCL:'L'      .BLKW    1      ; Device Type Class
E$IDTY:'L'      .BLKW    2      ; Device Type
E$IOPR:'L'      .BLKW    2      ; I/O Operation Count Longword
E$IERS:'L'      .BLKB    1      ; Soft Error Count
E$IERH:'L'      .BLKB    1      ; Hard Error Count

      .IF DF R$$MPL

E$IBLK:'L'      .BLKW    2      ; Blocks transferred count
E$ICYL:'L'      .BLKW    2      ; Cylinders crossed count

      .ENDC ; R$$MPL

      .EVEN

E$ILEN:'L'      ; Subpacket Length
;
; Flags for field E$IFLG
;
      EISSUB  ='B'          1 ; Subcontroller device
      .IF DF R$$MPL
      EISNUX  ='B'          2 ; No UCB extension, data invalid
      .ENDC ; R$$MPL

;
; Device Operation Subpacket
;
;
; +-----+
; | Device Operation Subpacket Length |
; +-----+
; | Task Name in RAD50 |
; +-----+
; | Task UIC |
; +-----+
; | Task TI: Logical Device Mnemonic |
; +-----+
; | Reserved | Task TI: Device Unit |
; +-----+
; | I/O Function Code |
; +-----+
; | Reserved | Operation Flags |
; +-----+
; | Transfer Operation Address |
; +-----+
; | Transfer Operation Byte Count |
; +-----+
; | Current Operation Retry Count |
; +-----+
;
;
;.=0

E$OLGN:'L'      .BLKW    1      ; Subpacket Length
E$OTSK:'L'      .BLKW    2      ; Task Name in RAD50
E$OUIC:'L'      .BLKW    1      ; Task UIC
E$OTID:'L'      .BLKB    2      ; Task TI: Logical Device Mnemonic
E$OTIU:'L'      .BLKB    1      ; Task TI: Logical Device Unit
      .BLKB    1      ; Reserved

```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```

E$OFNC:'L'      .BLKW  1      ; I/O Function Code
E$OFLG:'L'      .BLKB  1      ; Operation Flags
                  .BLKB  1      ; Reserved
E$OADD:'L'      .BLKW  2      ; Transfer Operation Address
E$OSIZ:'L'      .BLKW  1      ; Transfer Operation Byte Count
E$ORTY:'L'      .BLKW  1      ; Current Operation Retry Count

      .EVEN

E$OLEN:'L'      ; Device Operation Subpacket Length
;
; Flags for field E$OFLG
;
      E$TRA  ='B'      1      ; Transfer Operation
      E$DMA  ='B'      2      ; DMA Device
      E$EXT  ='B'      4      ; Extended Addressing Device
      E$PIP  ='B'     10      ; Device is positioning
;
; I/O Activity Subpacket
;
; +-----+
; | I/O Activity Subpacket Length |
; +-----+
;
;
;=0

E$ALGH:'L'      .BLKW  1      ; Subpacket Length
;
; I/O Activity Subpacket Entry
;
; +-----+
; | Logical Device Name Mnemonic |
; +-----+
; | Controller Number | Logical Device Unit |
; +-----+
; | Physical Subunit # | Physical Unit Number |
; +-----+
; | Physical Device Mnemonic (RSX-11M-PLUS only) |
; +-----+
; | Task TI: logical unit | Device flags |
; +-----+
; | Requesting Task Name in RAD50 |
; +-----+
; | Requesting Task UIC |
; +-----+
; | Task TI: Logical Device Name |
; +-----+
; | I/O Function Code |
; +-----+
; | Reserved | Flags |
; +-----+
; | Transfer Operation Address |
; +-----+
; | Transfer Operation Byte Count |
; +-----+
;
;=0

```

(continued on next page)

ERROR LOG PACKET FORMAT

Example C-1 (Cont.) Error Log Packet Format

```

E$ALDV:'L'      .BLKW  1      ; Logical Device Name Mnemonic
E$ALUN:'L'      .BLKB  1      ; Logical Device Unit
E$APCO:'L'      .BLKB  1      ; Controller Number
E$APUN:'L'      .BLKB  1      ; Physical Unit Number
E$APSU:'L'      .BLKB  1      ; Physical Subunit Number

      .IF DF R$SMPL

E$APDV:'L'      .BLKW  1      ; Physical Device Mnemonic

      .ENDC

E$ADFG:'L'      .BLKB  1      ; Device flags
E$ATIU:'L'      .BLKB  1      ; Task TI: Logical Unit
E$ATSK:'L'      .BLKW  2      ; Requesting Task Name in RAD50
E$AUIC:'L'      .BLKW  1      ; Requesting Task UIC
E$ATID:'L'      .BLKW  1      ; Task TI: Logical Device Name
E$AFNC:'L'      .BLKW  1      ; I/O Function Code
E$AFLG:'L'      .BLKB  1      ; Flags
                  .BLKB  1      ; Reserved
E$AADD:'L'      .BLKW  2      ; Transfer Operation Address
E$ASIZ:'L'      .BLKW  1      ; Transfer Operation Byte Count

      .EVEN

E$ALEN:'L'      ; Subpacket Entry Length
;
; Flags for field E$ADFG
;
      EA$SUB  ='B'          1  ; Subcontroller device
                  .IF DF R$SMPL
      EA$NUX  ='B'          2  ; No UCB extension, data invalid
                  .ENDC ; R$SMPL
;
; Flags for field E$AFLG
;
      EA$TRA  ='B'          1  ; Transfer Operation
      EA$DMA  ='B'          2  ; DMA Device
      EA$EXT  ='B'          4  ; Device has Extended Addressing
      EA$PIP  ='B'         10  ; Device is positioning

      .PSECT

      .MACRO  EPKDF$ X,Y
      .ENDM

      .ENDM

      .IIF NDF S$$YDF , .LIST

```


INDEX

ACTION statement, 5-30
/AP switch, 2-3, 2-8
Append switch, 2-3, 2-8
 Delete subswitch, 2-3, 2-8
Architecture
 control file module, 4-1
 error log control file, 4-1

/BA switch, 2-3, 2-9
Backup switch, 2-3, 2-9
BEGIN-END statement, 5-36
\$BMSET, 4-20
Brief format report, 3-11

CALDEV, 4-24
CALL statement, 5-33
CASE statement, 5-34
CDA, 1-1
CFL, 1-1, 1-3, 1-5, 5-1
 declaration
 conditional, 5-29 to 5-30
 data item, 5-26
 definition, 5-3
 scope, 5-26
 variable, 5-26
 definition, 5-2
 expression
 conditionals in, 5-2
 list handling, 5-39
 intermediate form modules,
 5-1
 lexical conditionals, 5-36
 signalling, 5-40
 spaces and tabs in text, 5-2
 statement
 ACTION, 5-30
 BEGIN-END, 5-36
 CALL, 5-33
 CASE, 5-34
 CONTROL, 5-32
 CRASH, 5-41
 DECLARE, 5-27
 DECODE, 5-32
 DECREMENT, 5-31
 DYNAMIC TABLE, 5-37
 ENABLE, 5-40
 FILE, 5-37 to 5-38
 FIND, 5-39
 IF-THEN-ELSE, 5-34
 INCREMENT, 5-31
 LEAVE, 5-35
 LIST, 5-39
 LITERAL, 5-33
 MESSAGE, 5-41
 MODULE, 5-32

CFL
 statement (Cont.)
 PACKET, 5-28
 POINTER, 5-38 to 5-39
 PRINT FORMATTING, 5-41
 PROCEDURE, 5-33 to 5-34
 PUT, 5-39
 RETURN, 5-33
 SEARCH, 5-40
 SELECT, 5-34 to 5-35
 SET, 5-31
 SIGNAL, 5-40 to 5-41
 SIGNAL STOP, 5-41
 SUBPACKET, 5-29
 TABLE, 5-37
 WHILE/UNTIL/DO, 5-35
 WRITE, 5-31
 WRITE GROUP, 5-31
 table structure, 5-36
 user interface
 command mode, 5-43
 option mode, 5-43
CFL command line
 DEVSM1, 4-36
CFL command mode, 5-43
CFL comments, 5-3
CFL data type
 ASCII, 5-5
 automatic conversion, 5-4
 binary byte, 5-4
 bit, 5-4, 5-7
 expression, 5-4
 logical, 5-4
 numeric, 5-5
 attribute option, 5-5
 byte, 5-5
 default, 5-5
 longword, 5-5
 quadword, 5-5
 radix option, 5-5
 value, 5-5
 word, 5-5
 numeric field, 5-7
 numeric literal, 5-6
 pointer, 5-7
 RSXTIME, 5-7
 string, 5-4
 numeric value, 5-4
 VMSTIME, 5-7
CFL expression, 5-9
 definition, 5-3
 logical operators, 5-10
 numeric operators, 5-13 to
 5-15

INDEX

CFL expression (Cont.)
 relational operators, 5-11 to 5-13
 string operators, 5-9
 CFL file
 command, 5-3
 error, 5-4
 input, 5-3
 report, 5-3
 user, 5-3
 CFL function
 %CND, 5-17
 %CNV, 5-17 to 5-18
 %CNV\$RSXTIME, 5-19
 %CNV\$STRING, 5-19
 %CNV\$VMSTIME, 5-19
 %COD, 5-19
 %COM, 5-20
 %CTL, 5-21
 format, 5-16
 %LOK, 5-21
 %PKT, 5-22
 %RPT, 5-22
 %STR, 5-23 to 5-24
 %TIM, 5-25
 %USR, 5-25
 CFL literals, 5-9
 CFL named variable, 5-8
 CONTEXT, 5-8
 COUNT, 5-9
 LENGTH, 5-8
 POINTER, 5-8
 CFL operands
 literals, 5-9
 variables, 5-9
 CFL operator precedence, 5-15
 CFL option mode, 5-43
 CFL primitives, 5-1
 CFL statement
 definition, 5-3
 %CND function, 5-17
 .CNF module, 4-1
 %CNV function, 5-17 to 5-18
 %CNV\$RSXTIME, 5-19
 %CNV\$STRING, 5-19
 %CNV\$VMSTIME, 5-19
 %COD, 5-19
 %COM, 5-20
 Compilation path
 RSX-11M, 4-11
 RSX-11M-PLUS, 4-12
 Compiler conditional
 literal declaration, 4-12
 Computational function, 5-20
 Concurrent I/O activity, 4-22, 4-24
 optional logging, 4-23
 Concurrent I/O error logging, 1-5
 Conditional declaration, 5-30
 Conditional function, 5-17
 Control file
 universal library, 5-1

Control File Language
 definition
 See also CFL, 4-2
 Control File Language compiler
 See also CFL
 Control file module, 1-3
 architecture, 4-1 to 4-2
 compilation path, 4-11
 CPU level module, 4-7
 data subpacket, 4-3
 definition, 4-1
 device activity subpacket, 4-3
 device ID subpacket, 4-3
 device operation subpacket, 4-3
 DEVSM1, 4-7
 DEVUDA, 4-7
 Dispatch, 4-3
 dispatching, 4-1
 DMPALL, 4-8
 DSP1M1, 4-4
 DSP1P1, 4-4
 DSP2M1, 4-5
 DSP2P1, 4-5
 DSP4M1, 4-5
 DSP4P1, 4-5
 DSP5M1, 4-5
 DSP5P1, 4-5
 DSP6M1, 4-6
 DSP6P1, 4-6
 DSP7M1, 4-6
 DSP7P1, 4-6
 E1134, 4-8
 E1144, 4-8
 E1160, 4-8
 E117X, 4-8
 E11XX, 4-8
 ERRORM, 4-7
 EUNKWN, 4-8
 FINLM1, 4-6
 FINLP1, 4-6
 FM1NM1, 4-6
 FM1WM1, 4-6
 FM2CM1, 4-6
 FM3CM1, 4-6
 FM4NM1, 4-6
 FM4WM1, 4-6
 FMTNP1, 4-7
 format, 5-1
 header subpacket, 4-3
 INITM1, 4-7
 interaction of interface, 4-13
 literal declaration, 4-12
 naming conventions, 4-1
 Non-DIGITAL device module, 4-1
 options, 4-12
 PARS1M, 4-4
 PARS2M, 4-4
 PARS3M, 4-4
 PARSEM, 4-4

INDEX

Control file module (Cont.)

- program control flow, 4-10
- PRS2AM, 4-4
- PRS2BM, 4-4
- recompilation, 4-13
- SELTML, 4-4
- SMRYEP, 4-7
- SMRYGP, 4-7
- SMRYHP, 4-7
- summary, 4-3
- task subpacket, 4-3
- CONTROL statement, 5-32
- CPU-level
 - dispatching module, 4-11
 - DSP5M1, 4-19
 - DSP5P1, 4-19
- CRASH statement, 5-41
- \$CRPKT, 4-23
- %CTL, 5-21
- /DA switch, 3-5, 3-8
- Date switch, 3-5, 3-8
- /DE switch, 3-5, 3-8
- DECLARE statement, 5-26 to 5-27
- DECODE statement, 5-32
- DECREMENT statement, 5-31
- Default fill character, 5-6
- Default print field width, 5-6
- Default radix, 5-6
- Defined report string
 - DIGITAL, 3-27
 - user, 3-28
- Device drivers
 - without error logging, 1-3
- Device error logging, 4-21 to 4-22
 - concurrent I/O activity, 4-22
- Device errors
 - error logging, 1-6
 - hardware register contents, 1-6
- Device I/O activity, 1-1
- Device information table, 4-34
 - ALTPRINTNAME field, 4-35
 - DISPNAME field, 4-35
 - DRIVETYPE field, 4-36
 - MASSBUSFLAG field, 4-36
 - MNEMONIC field, 4-35
 - PRINTNAME field, 4-35
 - SIZE field, 4-36
- Device module
 - EML11, 4-8
 - ERK05, 4-8
 - ERK67, 4-8
 - ERL12, 4-8
 - ERM05, 4-8
 - ERM23, 4-8
 - ERM80, 4-8
 - ERP07, 4-8
 - ERP23, 4-8
 - ERP456, 4-8
 - ERS11, 4-9
 - ERS34, 4-9

Device module (Cont.)

- ERX01, 4-9
- ERX02, 4-9
- ET0310, 4-9
- ET1645, 4-9
- ETAll, 4-9
- ETC11, 4-9
- ETS11, 4-9
- ETSV05, 4-9
- ETU58, 4-9
- ETU77, 4-9
- MSCP60, 4-9
- MSCP80, 4-9
- MSCPAT, 4-9
- MSCPE, 4-9
- MSCPEN, 4-9
- MSCPSD, 4-9
- MSCPTO, 4-9
- Device switch, 3-5, 3-8
- Device timeout, 4-20
- Device timeout logging, 4-21
- Device-level module, 4-11
 - adding to system, 4-25
 - addition to device
 - information table, 4-34
 - bit-to-text translation, 4-30
 - coroutine, 4-30
 - device data declaration, 4-26
 - device name, 4-26
 - error type, 4-30
 - EUNKWN, 4-37
 - exit, 4-31
 - * flag, 4-28
 - intermodule variable, 4-29
 - local work variable, 4-29
 - MASSBUS, 4-33 to 4-34
 - module statement, 4-26
 - Non-MASSBUS, 4-33 to 4-34
 - notes, 4-31
 - procedure statement, 4-26
 - register definitions, 4-27
 - subpacket statement, 4-26
- Devices
 - error logging, 2-4
 - control file module, 2-4 to 2-5
- DEVSM1, 4-7, 4-18, 4-34, 4-36
 - SLP, 4-36
- DEVUDA, 4-7
- Dispatch module, 4-3, 4-11
 - subpacket definition, 4-69 to 4-71
- Dispatch module path
 - brief report, 4-15
 - full report, 4-15
 - no report, 4-15
 - register report, 4-15
- Dispatch procedure, 4-10
- Dispatching
 - event-level, 4-16
- DMPALL, 4-8
- Drive serial numbers
 - significant numbers, B-1

INDEX

DSP1M1, 4-4
 subpacket definition, 4-72
 DSP1P1, 4-4
 subpacket definition, 4-72
 DSP2M1, 4-5, 4-36, 4-50 to 4-56
 subpacket definition, 4-73
 DSP2P1, 4-5, 4-36, 4-57 to 4-66
 subpacket definition, 4-73
 summary report, 4-16
 DSP3M1
 subpacket definition, 4-73
 DSP3P1
 subpacket definition, 4-73
 DSP4M1, 4-5
 subpacket definition, 4-74
 DSP4P1, 4-5
 subpacket definition, 4-74
 DSP5M1, 4-5, 4-19
 subpacket definition, 4-74
 DSP5P1, 4-5, 4-19
 subpacket definition, 4-74
 DSP6M1, 4-6
 subpacket definition, 4-74
 DSP6P1, 4-6
 subpacket definition, 4-74
 DSP7M1, 4-6
 subpacket definition, 4-75
 DSP7P1, 4-6
 subpacket definition, 4-75
 \$DTOER, 4-21
 \$DVCER, 4-21 to 4-22
 \$DVERR, 4-21 to 4-22
 \$DVTMO, 4-21
 \$DVTMO and \$DTOER, 4-20
 DYNAMIC TABLE statement, 5-37

 E1134, 4-8
 E1144, 4-8
 E1160, 4-8
 E117X, 4-8
 E11XX, 4-8
 ELI, 1-1
 /AP switch, 2-3, 2-8
 Append switch, 2-3, 2-8
 Delete subswitch, 2-3, 2-8
 /BA switch, 2-3, 2-9
 Backup switch, 2-3, 2-9
 error messages
 console terminal, 2-11
 user terminal, 2-11
 file naming switch, 2-8
 Hard error limit switch, 2-3,
 2-7
 /HL switch, 2-3, 2-7
 installing
 privileged, 2-1
 startup command file, 2-1
 invoking, 2-2
 privileged, 2-1
 /LIM switch, 2-3, 2-6
 Limit switch, 2-3, 2-6
 /LOG switch, 2-2 to 2-5, 2-8
 Log switch, 2-3 to 2-5, 2-8

ELI
 Log switch (Cont.)
 error limiting and, 2-2
 New version subswitch, 2-3,
 2-6
 No limit subswitch, 2-3,
 2-5 to 2-6
 No limit switch, 2-3, 2-6
 No log switch, 2-4
 /NOLIM switch, 2-3, 2-6
 /NOLOG switch, 2-4
 nonprivileged command, 2-2
 privileged commands, 2-2
 /R switch, 2-4
 /RE switch, 2-7
 Reset switch, 2-4, 2-7
 /SH switch, 2-4, 2-10 to 2-11
 Show switch, 2-2, 2-4, 2-10
 to 2-11
 /SL switch, 2-4, 2-7
 Soft error limit switch, 2-4,
 2-7
 subswitch summary, 2-3
 /SW switch, 2-4, 2-8
 switch functions, 2-1
 switch summary, 2-3
 Switch switch, 2-4, 2-8
 Delete subswitch, 2-4, 2-9
 New version subswitch, 2-4,
 2-9
 using ERRLOG defaults, 2-2
 ELI command
 reset limit, 1-4
 ELI show switch
 QIO count, 2-11
 ELI switches
 type
 display, 2-2
 file naming, 2-2
 limiting, 2-2
 logging, 2-2
 EML11, 4-8
 ENABLE statement, 5-40
 Encoding function, 5-19
 Entry procedure, 4-10
 EPKDF.MAC, 4-16
 ERK05, 4-8
 ERK67, 4-8
 ERL12, 4-8
 ERLCFL Report Messages, 5-44
 ERLCNF Report Messages, 3-29
 fatal, 3-29
 information, 3-34
 ERLRPT Report Messages, 3-35
 control file, 3-35
 ERM05, 4-8
 ERM23, 4-8
 ERM23 device-level module, 4-37
 to 4-49
 ERM80, 4-8
 ERP07, 4-8
 ERP23, 4-8
 ERP456, 4-8

INDEX

- ERRDEFINE.CFS, 3-27
- ERRLOG, 1-1
 - Backup file, 2-9
 - backup file, 1-4
 - defaults, 2-2
 - ELI commands, 1-3 to 1-4
 - installing
 - privileged, 2-1
 - log file, 1-4
 - mandatory installation, 2-1
- ERRLOG Messages, 2-12
- ERRLOG.ULB, 3-2
 - module addition, 4-36
 - tuning, A-1
- ERRLOGETC.ULG, 4-36
- Error limit, 1-4
 - device mount, 2-6
 - logging and, 2-6
 - notification, 1-4
 - reset, 1-4, 2-7
 - reset counts, 2-6
 - system reboot, 2-6
- Error Log
 - Interface
 - See also ELI
- Error log
 - command packet, 1-4
 - control file, 1-3
 - definition, 4-1
 - devices, 2-4
 - file, 1-1
 - definition, 4-2
 - format, 5-1
 - naming, 2-8
 - RPT, 1-3
 - packet, 1-1
 - command, 1-3
 - creation, 4-23
 - definition, 4-2
 - ERRSEQ, 1-3
 - format, 1-4, 5-1, C-1
 - packet number
 - definition, 1-3
 - processing, 5-1
 - queue, 4-24
 - remove from queue, 4-25
 - SMSG\$, 1-4
 - structure, 4-3
 - unsolicited interrupt, 4-23
 - packet number, 3-1
 - report
 - context information, 1-5
 - subpacket
 - definition, 4-2
 - system
 - task interaction, 1-2
- Error log c
 - ontrol file
 - architecture, 4-1
- Error Logger
 - See also ERRLOG
- Error logging
 - device errors, 1-6
- Error logging (Cont.)
 - executive support, 1-6
 - device errors, 1-6
 - device timeouts, 1-6
 - memory errors, 1-6
 - unexpected traps or
 - interrupts, 1-6
 - interrupt timeouts, 1-6
 - memory errors, 1-6
 - support on Non-DIGITAL device,
 - 4-19
 - device level module, 4-19
 - driver, 4-19
 - universal library entry,
 - 4-19
 - Unexpected traps or
 - interrupts, 1-6
 - unknown device
 - See EUNKWN
 - unused vectors, 1-6
- Error logging devices
 - control file module, 2-4 to
 - 2-5
- Error Logging System
 - hard error, 1-3
 - soft error, 1-3
- Error logging system
 - CDA, 1-1
 - Control file language
 - compiler, 1-1
 - device I/O activity, 1-1
 - Error log interface, 1-1
 - Error logger, 1-1
 - executive routines, 1-1
 - mass storage device errors,
 - 1-1
 - memory errors, 1-1
 - operating system failure, 1-1
 - Report generator, 1-1
- ERROR module, 4-19
- Error sequence number
 - See ERRSEQ
- Error type definition, 1-5
- ERRORM, 4-7
- ERRSEQ
 - Append switch, 1-3
 - change, 1-3
 - RMD use, 1-3
- ERS11, 4-9
- ERS34, 4-9
- ERX01, 4-9
- ERX02, 4-9
- ET0310, 4-9
- ET1645, 4-9
- ETAll, 4-9
- ETC11, 4-9
- ETS11, 4-9
- ETSV05, 4-9
- ETU58, 4-9
- ETU77, 4-9
- EUNKWN, 4-8, 4-19, 4-37
- Event
 - definition, 1-3, 4-2

INDEX

- Event-level dispatching, 4-16
 - control information, 4-17
 - CPU-detected errors, 4-17
 - device control information, 4-17
 - device errors, 4-17
 - device information, 4-17
 - dispatcher module, 4-18
 - error log control code, 4-17
 - event code, 4-18
 - event type, 4-18
 - format, 4-18
 - system control information, 4-17
- Example
 - Brief format report, 3-13 to 3-14
 - device information table, 4-34
 - DSP2M1 dispatch module, 4-14
 - DSP2M1 dispatcher module, 4-50 to 4-56
 - DSP2P1, 4-15
 - DSP2P1 dispatcher module, 4-57 to 4-66
 - ERM23, 4-26
 - ERM23 device-level module, 4-37 to 4-49
 - ERM23 module, 4-14
 - ERM23 notes file, 4-31
 - error log packet format, 5-2
 - error log subpacket format, 5-2
 - Full format report, 3-16 to 3-17
 - notes module, 4-67 to 4-68
 - NRM23 notes module, 4-67 to 4-68
 - Register report, 3-19
 - RM02/03
 - register printing, 4-27
 - RM02/03 module, 4-38 to 4-49
 - RM03 module, 4-13
- Example ERM23 module, 4-15
- Executive error logging support, 1-6
- Executive routines
 - error collection
 - device registers, 1-3
- Executive Send Message
 - directive
- See SMSG\$
- /F switch, 3-5
- FILE statement, 5-37 to 5-38
- FIND statement, 5-39
- FINLM1, 4-6
- FINLP1, 4-6
- FM1NM1, 4-11
- FM1WM1, 4-6, 4-11
- FM2CM1, 4-6, 4-11
- FM3CM1, 4-6, 4-11
- FM4NM1, 4-6
- FM4WM1, 4-6, 4-11
- FMTNP1, 4-7, 4-11
- FMTPL, 4-11
- FN1NM1, 4-6
- \$FNERL, 4-22
- Format module, 4-11
- Format switch, 3-5
 - full report, 3-15
 - no report, 3-19
- Register report, 3-18
- Full format report, 3-15
 - context information, 1-5
 - device error, 1-5
 - device supplied information, 1-5
 - * flag, 1-5
 - I/O operation, 1-5
- Hard error
 - definition, 1-3
- Hard error limit switch, 2-3, 2-7
- /HL switch, 2-3, 2-7
- .ICF module, 4-1
- IF-THEN-ELSE statement, 5-34
- INCREMENT statement, 5-31
- INITM1, 4-7
- Installation
 - RPT, 3-1
- Intermodule variable, 4-29
 - block number, 4-29
 - cylinder error, 4-29
 - device function, 4-29
 - drive serial number, 4-29
 - drive type, 4-29
 - error type, 4-29 to 4-30
 - group error, 4-29
 - head error, 4-29
 - MASSBUS, 4-33
 - Non-MASSBUS, 4-33
 - physical unit number, 4-29
 - sector error, 4-29
- Intermodule variable
 - declaration, 4-14
- Interrupt timeouts
 - error logging, 1-6
- LBR
 - module name requirement, 4-17
- LBR command line
 - device-level module, 4-37
 - DEVSM1, 4-36
 - DSP2M1, 4-36
 - DSP2P1, 4-36
- LEAVE statement, 5-35
- Lexical conditionals, 5-36
- /LIM switch, 2-3
- /LIM switch, 2-6
- Limit switch, 2-3, 2-6
- LIST statement, 5-39
- Literal declaration, 4-12
- LITERAL statement, 5-33

INDEX

/LOG switch, 2-3, 2-5, 2-8
 Log switch, 2-3, 2-5, 2-8
 New version subswitch, 2-3, 2-6
 No limit subswitch, 2-3, 2-5 to 2-6
 \$LOGER, 4-23
 LOGTST
 ERROR routines, 4-23
 %LOK, 5-21
 Lookahead function, 5-21
 .LST module, 4-1
 LUN
 calculation, 4-24
 LX:, 3-2

 Mass storage device errors, 1-1
 MCR SSM command
 See SSM command, 1-3
 Memory errors, 1-1
 error logging, 1-6
 pregenerated systems, 1-6
 MESSAGE statement, 5-41
 Messages
 ERLCFL, 5-44
 Mixed MASSBUS configuration, 4-33
 MODULE statement, 5-32
 MSCP60, 4-9
 MSCP80, 4-9
 MSCPAT, 4-9
 MSCPE, 4-9
 MSCPEN, 4-9
 MSCPSD, 4-9
 MSCPTO, 4-9

 Named variable, 5-8
 NML11, 4-9
 No limit switch, 2-3, 2-6
 No log switch, 2-3
 /NOLIM switch, 2-3, 2-6
 /NOLOG switch, 2-3
 Nonsense interrupt logging, 4-22
 Notes module
 exit, 4-33
 heading, 4-32
 MODULE statement, 4-32
 NML11, 4-9
 note name, 4-32
 NRK67, 4-9
 NRM05, 4-9
 NRM23, 4-9
 NT0310, 4-9
 NTS11, 4-9
 print declaration, 4-32
 print number, 4-33
 PROCEDURE statement, 4-32
 unknown note, 4-33
 user written, 4-31
 NRK67, 4-9
 NRM05, 4-9
 NRM23, 4-9

 \$NSIER, 4-22
 NT0310, 4-9
 NTS11, 4-9
 Numeric conversion function, 5-17 to 5-18

 Operating system failure, 1-1

 /P switch, 3-5, 3-8
 Packet information function, 5-22
 Packet number, 3-1
 Packet number switch, 3-5, 3-8
 Packet selection switches, 3-8
 PACKET statement, 5-28
 PARS1M, 4-4
 PARS2M, 4-4
 PARS3M, 4-4
 PARSEM, 4-4
 %PKT, 5-22
 POINTER statement, 5-38 to 5-39
 PRINT FORMATTING statement, 5-41
 keyword string, 5-41 to 5-42
 PROCEDURE statement, 5-33 to 5-34
 Processor priority, 4-20
 PRS2AM, 4-4
 PRS2BM, 4-4
 PUT statement, 4-31, 5-39

 \$QERMV, 4-25
 QUPKT, 4-24

 /R switch, 2-4, 2-7, 3-6, 3-27
 Register report, 3-18
 Report control function, 5-22
 Report Generator
 See also RPT
 Report switch, 3-6, 3-27
 Reset switch, 2-4, 2-7
 RETURN statement, 4-31, 5-33
 RM02/03 module, 4-37 to 4-49
 RM02/03 notes module, 4-67 to 4-68
 %RPT, 5-22
 RPT, 1-1
 command line, 3-2
 input file, 3-2
 multiple qualifiers, 3-3
 report file, 3-2
 switches, 3-2
 universal library, 3-2
 concurrent I/O error logging, 1-5
 /DA switch, 3-5, 3-8
 Date switch, 3-5, 3-8
 /DE switch, 3-5, 3-8
 Device switch, 3-5, 3-8
 multiple qualifiers, 3-4
 error log control file, 1-4
 error log file, 1-4
 Error log packet

INDEX

RPT

Error log packet (Cont.)
 number, 3-1
 error log packets, 1-4
 error type definition, 1-5
 /F switch, 3-5, 3-11
 Format switch, 3-5, 3-11
 brief report, 3-11
 full report, 3-15
 No report, 3-19
 Register report, 3-18
 information required, 3-2
 format selection, 3-2
 packet selection, 3-2
 summary selection, 3-2
 interpreter, 5-1
 /P switch, 3-5, 3-8
 Packet number switch, 3-5,
 3-8
 multiple qualifiers, 3-4
 /R switch, 3-6, 3-27
 report information selection,
 1-4 to 1-5
 Report switch, 3-6, 3-27
 /SE switch, 3-6, 3-9
 Serial number switch, 3-6,
 3-9
 multiple qualifiers, 3-4
 /SU switch, 3-6, 3-20
 subswitch summary, 3-5
 Summary switch, 3-6, 3-20
 error report, 3-20
 geometry report, 3-23
 multiple qualifiers, 3-4
 no report, 3-27
 summary switch
 history report, 3-25
 switch summary, 3-5
 /T switch, 3-7, 3-9
 Type switch, 3-7, 3-9
 multiple qualifiers, 3-4
 /V switch, 3-7, 3-10
 Volume label switch, 3-7,
 3-10
 /W switch, 3-7, 3-28
 Width switch, 3-7, 3-28
 RPT control function, 5-21
 RPT default command line, 3-4
 Date switch, 3-4
 Device switch, 3-4
 Format switch, 3-4
 Packet switch, 3-4
 Summary switch, 3-4
 Type switch, 3-4
 Width switch, 3-4
 RPT default file specification,
 3-3
 RPT report switch
 format packet, 3-5
 select packet, 3-5
 summarize packet, 3-5
 RPTBLD.BLD file, 3-2
 RSXTIME, 5-7

/SE switch, 3-6, 3-9
 SEARCH statement, 5-40
 SELECT statement, 5-34 to 5-35
 SELTM1, 4-4
 Serial number switch, 3-6, 3-9
 SET statement, 5-31
 /SH switch, 2-4, 2-10 to 2-11
 /Show switch, 2-10 to 2-11
 Show switch, 2-4
 SIGNAL statement, 5-40 to 5-41
 SIGNAL STOP statement, 5-41
 Signalling
 definition, 5-40
 /SL switch, 2-4, 2-7
 SMRYEP, 4-7
 SMRYGP, 4-7
 SMRYHP, 4-7
 SMSG\$, 1-4
 Soft error
 definition, 1-3
 Soft error limit switch, 2-4,
 2-7
 SSM command, 1-3
 %STR, 5-23 to 5-24
 String conversion function,
 5-19
 String handling function, 5-23
 to 5-24
 /SU switch, 3-6, 3-20
 Subpacket definition, 4-69
 dispatch module, 4-69 to 4-71
 DSP1M1, 4-72
 DSP1P1, 4-72
 DSP2M1, 4-73
 DSP2P1, 4-73
 DSP3M1, 4-73
 DSP3P1, 4-73
 DSP4M1, 4-74
 DSP4P1, 4-74
 DSP5M1, 4-74
 DSP5P1, 4-74
 DSP6M1, 4-74
 DSP6P1, 4-74
 DSP7M1, 4-75
 DSP7P1, 4-75
 SUBPACKET statement, 5-29
 Summary switch, 3-6, 3-20
 all report, 3-20
 error report, 3-20
 geometry report, 3-23
 history report, 3-25
 no report, 3-27
 /SW switch, 2-4, 2-8
 Switch switch, 2-4, 2-8
 Delete subswitch, 2-4, 2-9
 New version subswitch, 2-4,
 2-9
 System Service Message command
 See SSM command
 /T switch, 3-7, 3-9
 Table
 definition, 5-36

INDEX

Table (Cont.)	USERCM, 3-2
RPT use, 5-36	%USR, 5-25
TABLE statement, 5-37	
%TIM, 5-25	/V switch, 3-7
Time handling function, 5-25	VMSTIME, 5-7
Type switch, 3-7, 3-9	Volume label switch, 3-7
Unexpected traps or interrupts, 1-6	/W switch, 3-7, 3-28
User defined switch string, 3-28	WHILE/UNTIL/DO statement, 5-35
User I/O function, 5-25	Width switch, 3-7, 3-28
	WRITE GROUP statement, 5-31
	WRITE statement, 4-30, 5-31

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____

Zip Code _____
or Country _____

Do Not Tear - Fold Here and Tape

digital



No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061



Do Not Tear - Fold Here

Cut Along Dotted Line